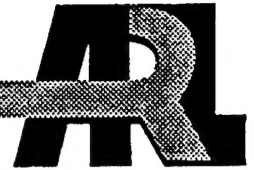ARMY RESEARCH LABORATORY

Texture Generation
for use in Synthetic Scenes

By
Max P. Bleiweiss
Battlefield Environment Directorate

and
Clem Z. Ota and J. Michael Rollins
Science and Technology Corporation

DTIC QUALITY INSPECTED 2

ARL-TR-1015   **March 1997**

19970827 094

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The citation of trade names and names of manufacturers in this report is not to be construed as official Government indorsement or approval of commercial products or services referenced herein.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE March 1997	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE
Texture Generation for use in Synthetic Scenes

5. FUNDING NUMBERS

6. AUTHOR(S)
M. Bleiweiss, C. Ota, J. M. Rollins

7. PERFORMING ORGANIZATION NAMES(s) AND ADDRESS(ES)
U.S. Army Research Laboratory Directorate

ATTN: AMSRL-BE-S
White Sands Missile Range, NM 88002-5504

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-1015

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
Adelphi, MD 20783-1145

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER
ARL-TR-1015

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (Maximum 200 words)

In the process of creating synthetic scenes for use in simulations/visualizations, texture is used as a surrogate to "high" spatial definition. For example, if one were to measure the location of every blade of grass and all of the characteristics of each blade of grass in a lawn, then in the process of composing a scene of the lawn, it would be expected that the result would appear "real"; however, because this process is excruciatingly laborious, various techniques have been devised to place the required details in the scene through the use of texturing. Experience gained during the recent Smart Weapons Operability Enhancement Joint Test and Evaluation has shown the need for higher fidelity texturing algorithms, and a better parameterization of those that are in use. In this study, four aspects of the problem have been analyzed: texture extraction, texture insertion, texture metrics, and texture creation algorithms. The results of extracting real texture from an image, measuring it with a variety of metrics, and generating similar texture with three different algorithms is mixed. The metric (correlation length) thought to be not very capable as a seed for generating texture (based on experience during the evaluation) seems to produce the best results. And though best, the ability is still lacking. The other two metrics are fractal dimension and several parameters derived from the grey-level co-occurrence matrix. These same metrics can be used to define clutter and to make comparisons between "real" and synthetic (or artificial) scenes in an objective manner.

14. SUBJECT TERMS
texture, synthetic scene, SWOE, IR

15. NUMBER OF PAGES
177

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT SAR

Preface

The results of comparing real and synthetic texture during the recent Smart Weapons Operability Enhancement Joint Test and Evaluation suggested that a better way of generating texture for placement in synthetic scenes be developed. In this study, we investigate three accepted measures of texture: correlation length, fractal dimensions, and parameters derived from the grey-level co-occurrence matrix as candidates for a different texture generation algorithm. The output of these measures (or metrics) are used as input to three texture generating algorithms. A comparison is then made between the input and output textures. The results of this study were not definitive and future work is indicated. This study was supported by the Smart Weapons Operability Enhancement Joint Test and Evaluation Program Office, Dr. J.P. Welsh, director.

Contents

Figures

4

Tables

Executive Summary

In the process of creating synthetic scenes for use in simulations/visualizations, texture is created as a surrogate for high spatial definition. For example, measuring the location and characteristic of every blade of grass in a lawn, then simulating a scene of it would be excruciatingly laborious. Various techniques have been devised to place the required details in the scene through the use of texturing. Experience gained during the recent Smart Weapons Operability Enhancement Joint Test and Evaluation (SWOE JT&E) has shown the need for higher fidelity texturing algorithms, and a better parameterization of those that are in use. This study, analyzes four aspects of the problem: texture metrics, texture creation algorithms, texture extraction, and texture insertion.

The overall idea is to see if a textural property can be measured with a metric, and use it as a seed for the creation of texture with that same textural property. Textural metrics can be parameterized, resulting in texture created for insertion into a scene. Texture extraction is the problem of determining metrics capable of capturing the textural aspects of a scene element. Texture insertion is just the reverse problem; mapping a texture into a scene element. Furthermore, if the texture metrics are parameterized, the task of synthetic scene generation is simplified. For example, if a particular metric describes the texture of a certain scene element and that metric varies only with time of day, then that parameter (time), can be used to determine what the texture should measure at that time. The question becomes, can we generate a texture that measures properly according to some metric? The major purpose of this study is to determine the algorithms that use certain texture metrics as seeds, and to generate texture maps. These maps are studied to determine if they yielded correct measures of their texture. The three texture metrics used in this study are those used in the SWOE JT&E analyses: correlation length, metrics derived from the grey level co-occurrence matrix, and fractal dimension. Reviewers of this report noted that some of our fractal dimensions are not of the expected value. The textured metrics used are limited in their use and application. We did not attempt to study this situation in detail, although it is addressed in the conclusions of the report. The texture generation algorithms used in this study are based on the same three metrics.

1. Introduction

Haralick and Shapiro define texture as the property of an image where small regions of the image experience "wide variation of tonal features." [1] This is opposed to the grey-tone property where small regions experience "little variation of discrete tonal features." The small regions make up the set of feature and background fragments of an image, which can be resolved as unique portions of the image. The textural properties are below the resolution threshold for definition of features. A grey-tone feature is a region of contiguous pixels that are all at or near the same grey level: all white, all black, or somewhere in between. A textural feature is a region of contiguous pixels whose grey level varies from pixel to pixel, or from a couple of nearby pixels to another couple of nearby pixels. The variation takes on a continuum of characteristics from a variety of highly-ordered, or structured, patterns to a variety of random patterns. The textural features can be composed of large, grey-tone features; small, grey-tone features or a combination. The measures, or metrics, are the tools used to describe texture and are based on a variety of mathematical operators that are applied to the 2-D array of pixels. The real-world properties that cause texture to exist are relief, and optical and thermal characteristics (such as emissivity, thermal conductivity of the various scene elements), which exist on a small scale relative to the rest of the elements (features) in the scene.

This paper addresses how the fine detail, which is present in the real world, be represented in synthetic imagery without resorting to the laborious and time-consuming efforts of including the details in a modeling and simulation effort.

Figure 1 provides an example of high-resolution texture map. This image was obtained during the SWOE JT&E [2] by the Waterways Experiment Station (WES) of the U.S. Army Corps of Engineers. It is a high spatial resolution image of a small region of natural terrain located at the Yuma I field test site (one of the locations used during the SWOE JT&E series of field tests). The data was obtained with a far infrared (IR) (8 to 12 μm), thermal imager configured with a narrow field-of-view lens. The texture images provide the raw material for the SWOE JT&E synthetic texture generation approach to the problem of creating fine-detail synthetic texture without time-consuming, highly-detailed modeling.
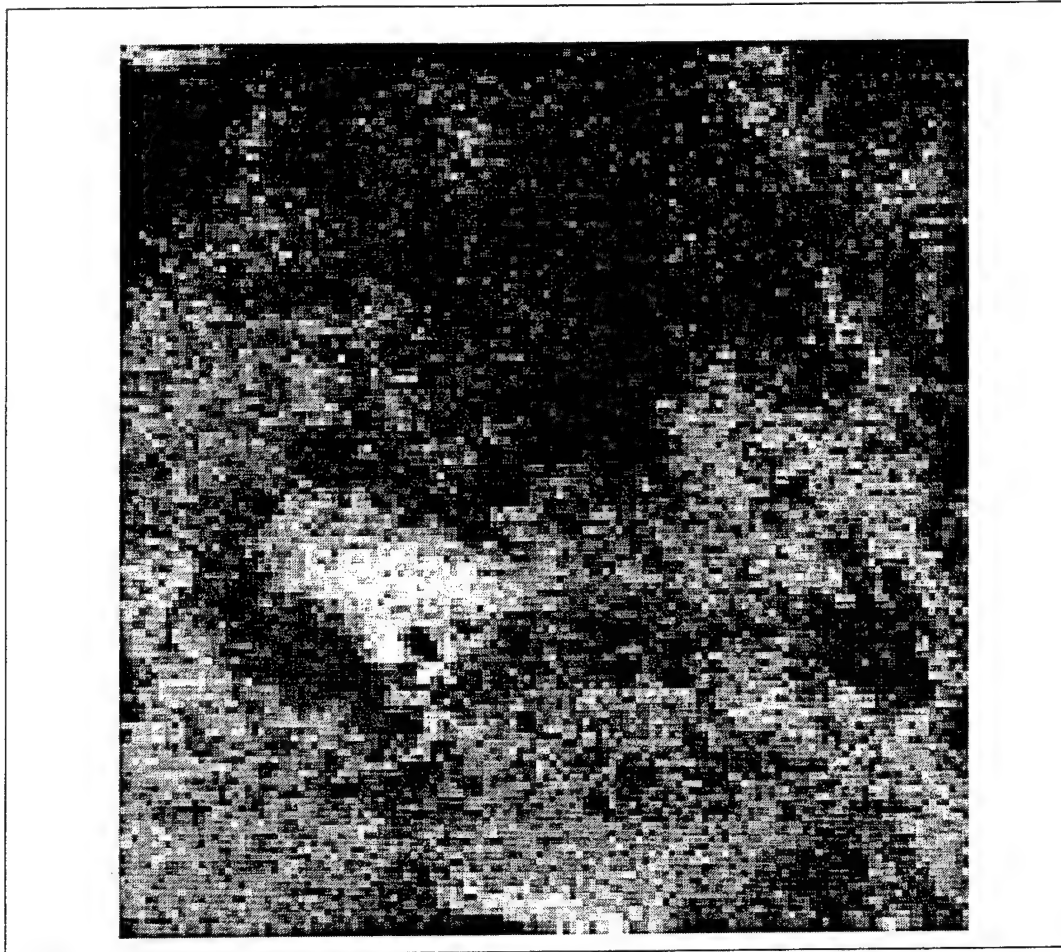
Figure 1. An example of a texture image, obtained during the SWOE JT&E to define a texture field for input to the SWOE image generating process.

Textural measures used to compare the real and synthetic imagery indicate that the procedures for introducing texture into a synthetic image are flawed, in some cases as indicated by the results of the analyses of the SWOE JT&E synthetic imagery. [3] Figure 2 illustrates this problem. The figure consists of three panels: a real image, a synthetic image, and a homogeneous region map. The variation in grey level within homogeneous regions (defined approximately by the panel on the right) in the synthetic image is generally lacking. There are artifacts in the various panels that should be ignored for the purposes of the discussion here, such as the relative contrast between the trees and terrain for real and synthetic images, and the border around the real image.
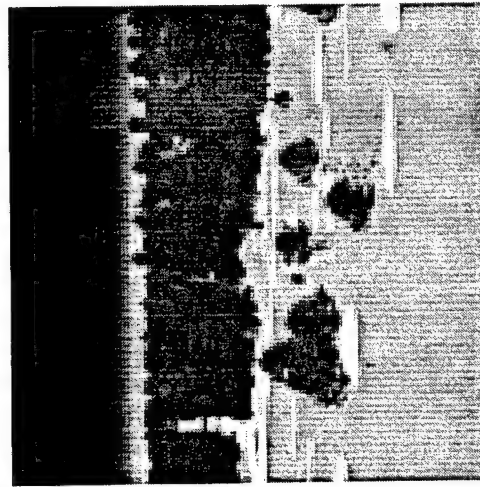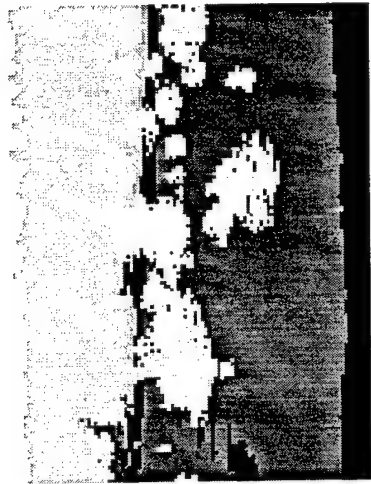
homogeneous region definition

synthetic image

real image

Figure 2. Three panels demonstrate the presence of texture in the real scene.

Developing texture-generating algorithms and comparing the resulting synthetic textures among themselves, and with real texture is the focus of this paper. The textural metrics and the texture generating algorithms used in this investigation are briefly discussed. A comparison is made between the output of the algorithms and the real textures.

2. Texture Metrics

Texture metrics, or measures, are used to quantify a textural property of the region of interest so that various regions may be compared to see if they contain similar or different textures. For example, in the SWOE JT&E Final Report [2] textural metrics were used to determine if the same portions of the real and the synthetic image were actually the same. The standard, first-order metrics, such as average grey level, do not give any information as to the textural structure of the image. The second order metrics, such as correlation length do allow comparison of textures between images. These metrics become the eyes that allow descriptive comments such as herring-boned structure, fine-grained, coarse-grained, random, and isotropic to be made about a textural feature. These metrics can also be used to segment an image into regions of similar texture through application of a particular metric to small regions of the image. These metrics are rarely used to actually compare regions for the determination of a degree of similarity because there are no standards for these quantities to rank the texture feature under study. For example, a particular feature is 0.9 out of 1.0 of having the particular property being measured.

The three measures discussed in this paper are:

Correlation length: The lag where the spatial autocorrelation declines to 1/e of its maximum value. Lag, in this instance, is the distance measure defined by the sampling interval in space, such as a pixel.

A derivative of the grey-level co-occurrence matrix (GLCM): The GLCM measures "the dependence between pairs of grey levels arising from pixels in a specified spatial relation. " [1] For example, the number of times a given grey-level pair occur side-by-side, relative to one another, at separations of two pixels, three pixels, and so on is examined. A variety of metrics (entropy, contrast, etc.) based on various moments of the GLCM have been developed that provide varying degrees of parameterization of the texture.

Fractal dimension: For the purposes of this paper, the fractal dimension is defined as the parameter D in the following equation:

$$D = n + 1 - H \qquad (1)$$

with $n = 2$, the dimension of the process (a line has a value of 1, a two-dimensional process, 2, and a three-dimensional process, 3), and

$$2H = \beta - n. \qquad (2)$$

Under the assumption that the image grey-level variation can be modeled as a "fractional Brownian 'noise'" ($0 < H < 1$), the value of β is determined from the slope of the power spectral density function of the image. The fractal dimension then measures the degree of roughness of the image. [4] [5]

These metrics are used to compare various synthetic textures. These metrics can also be used as "seeds" for algorithms to generate texture.

There is some question as to some of the results of applying these three metrics to a particular sample of data. In particular, some of the fractal dimensions we arrive at are less than 2.0. According to the literature, this would indicate a line and not a surface. All the metrics we use, and many others as well, do not behave predictably all of the time for a variety of reasons. In the case of the GLCM, the way an image is quantized (eight bits versus four bits, or even two bits) results in differing GLCMs. There is no guidance for choosing one over the other. There are several ways to define correlation length. Hilgers et al. compare different measurements of correlation length and assess the stability and precision of their results. [6] They conclude that their results depend on how the correlation length is computed, which depends on how the image is sampled, and the finite size of the sample.

As we compute it, the fractal dimension depends on the measurement of the slope of the power spectral density curve, which depends on the spectral leakage caused by the windowing function, the presence of trends, and aliasing. This says nothing about the assumption that power-law scaling exists. Some of these issues have been addressed by Austin et al. and Fox. [7] [8] Fox states that the Fourier analyses has a strong statistical foundation behind its application that is often overlooked. On the other hand, the fractal analysis does not, and it appears as though this is still true today. Fox executes an empirical study between fractal dimension and power-law

frequency spectra to conclude that the relationship is non-linear, except in the region near $\beta = 2$, which is also the region where the slopes cluster due to spectral leakage. [7] This does not necessarily solve the problem of questionable numbers, but serves to show that much remains to be accomplished in this field and that a more in-depth analysis should be performed along the lines of that Fox demonstrates.

3. Texture Algorithms

A texture map, generated from a fractal dimension algorithm is expected to produce a measure that is the same value as the fractal dimension. If that measure were capable of capturing all aspects of the real texture and, at the same time, the texture-generating algorithm based on that measure was a capable algorithm, the other textural metrics (correlation length, etc.) would also be expected to provide the same measure for the real and synthetic textures. If some aspect of this expectation is not met, then further study will be suggested. It is not expected that any one of these algorithms will be successful 100 percent of the time. Therefore, we chose three measures to use in three different algorithms to generate texture.

The mid-point displacement algorithm is a method of generating a two-dimensional fractional Brownian motion or noise. This approximate method is described in *The Science of Fractal Images*. [9] The mid-point displacement algorithm produces an image that only approximates a fractional Brownian "motion". Another algorithm, based on this same concept, does a better job. [10] See appendix A.

The GLCM method for building the textures presented here, is based on the algorithm described in the paper by G. Lohmann. [11] In the algorithm, a set of four grey-level co-occurrence matrices provide feature vectors. Lohmann reports that the four primary co-occurrence matrices (horizontal, vertical, and left- and right-diagonal) contain sufficient information to synthesize texture that closely resemble textures from remotely-sensed images of the Landsat/TM and ERS-1/AMI satellite-borne sensors. Our idea is to determine how well this algorithm performs on our type of imagery.

For a description of the correlation length algorithm used in this study, see appendix G where the basic methodology is laid out: [12]

> Numerous approaches to texturing were reviewed and a simplified two-dimensional, autoregressive (AR) model was selected. It uses correlation length in vertical and horizontal directions and brightness mean and standard deviation as input parameters for a kernel (process of order [1,1]). The model was

developed from an AR model used in the U.S. Air Force Infrared Modeling and Analysis (IRMA) image modeling system.

An empirical approach was taken because of the lack of general theory or models on thermal IR texture. Using this approach homogeneous surfaces (grass, bare soil, trees, tree lines, etc.) of interest were imaged for the times (or under similar conditions) the synthetic scenes were to be generated. Textural features of these surfaces were measured and input to an AR texture generator program, which generated isotropic Gaussian texture maps. These maps were used by the rendering software to texture the polygons.

The raw output of the basic program, the isotropic Gaussian texture maps, is smoothed to reduce vertical and horizontal striation artifacts for long correlation lengths. Both raw and smoothed maps are used in the study reported here.

Figure 3 shows the target texture attempted to be reproduced by the various algorithms discussed in this section. The map is a 32 by 32 image (segmented from the image/texture map of figure 1), which has texture properties that are given in table 1.
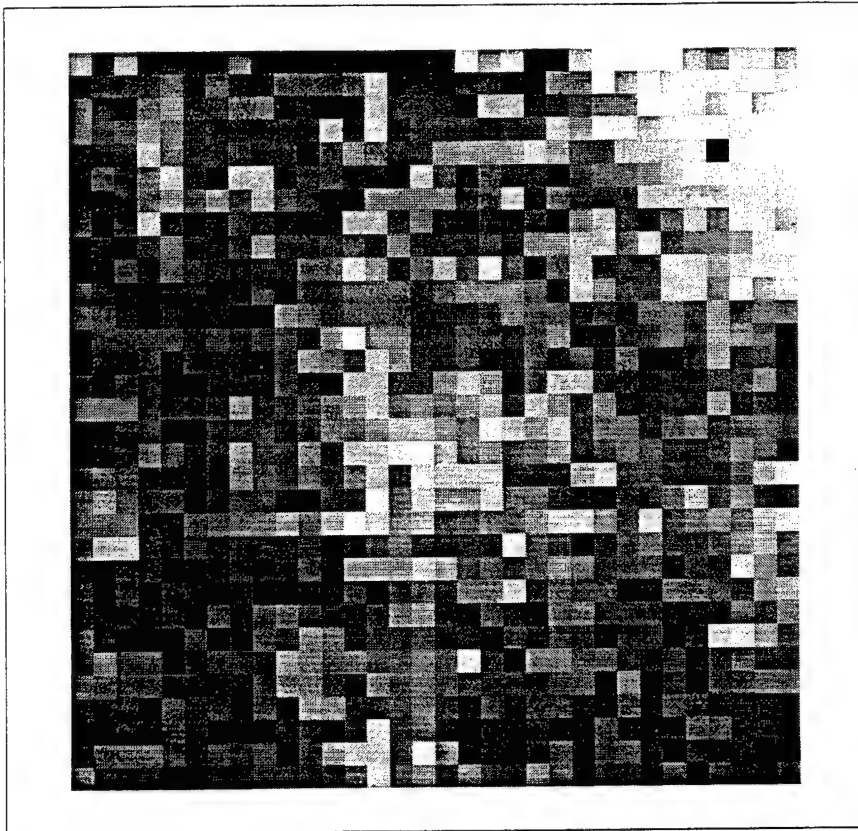
Figure 3a. Texture that is the target for the various generating algorithms.
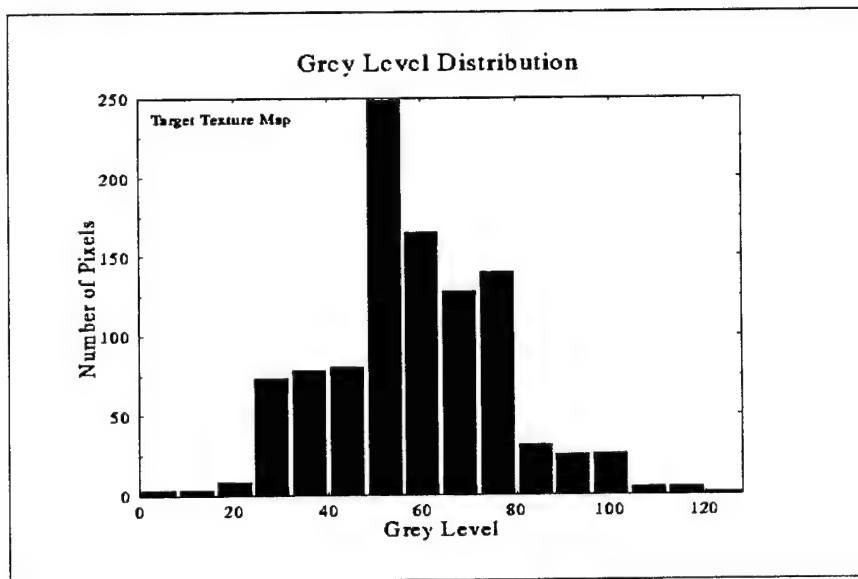
Figure 3b. The histogram of grey levels for the target texture image of figure 3a.

The results shown in table 1 are derived from the application of the corresponding metric to the image. The table headings are acronyms for the metrics discussed briefly in section 2. With the exception of the mean and variance, the metrics measure the second-order statistics of the images. All have been used in studies of the textural properties of images. More details on the form and application of these metrics can be found in Bleiweiss et al. [3]

In tables 1, 2 and 3 MEAN is the average grey level of the texture map (ranges from 0 to 128), VAR is the variance of the grey levels in the texture map, FRAC D is the fractal dimension, ACL is the autocorrelation length, CONTR, CORR, ENTRPY, and HOMOG, are the GLCM metrics called contrast, correlation, entropy, and homogeneity, respectively. The GLCM metrics are somewhat self-explanatory. More detailed explanations of these metrics are in appendix C.

Table 1. Texture measure values for the target texture

Mean	VAR	FRAC D	ACL	CONTR	CORR	Entropy	HOMOG
57.247	338.116	1.965	3.537	222.219	0.667	4.644	0.014

3.1   Mid-Point Displacement Algorithm

The mid-point displacement algorithm is a method of generating a two-dimensional fractional Brownian motion or noise. This approximate method is described in *The Science of Fractal Images*. [9] Quoting from this reference, but modifying the discussion to fit the two-dimensional aspects of our problem, a two-dimensional fractional Brownian motion is defined as a two-dimensional process (a random field) $X(t_1, t_2)$ with the following properties:

> The increments $X(t_1, t_2) - X(s_1, s_2)$ are Gaussian with zero mean and the variables s and t are positions in the process; i.e., t is the x-y coordinate of the point under discussion and s is another point at another x-y coordinate at some distance removed from t.

The variance of the increments $X(t_1,t_2) - X(s_1,s_2)$ depends only on the distance

$$\sqrt{\sum_{i=1}^{2}(t_i - s_i)^2}$$

and is proportional to the $2H -$ th power of the distance, where the parameter H again satisfies $0 < H < 1$. Thus,

$$E\left(\left|X(t_1,t_2) - X(s_1,s_2)\right|^2\right) \propto \left(\sum_{i=1}^{2}(t_i - s_i)^2\right)^H$$

or

$$\mathrm{Var}\left(X(t_1,t_2) - X(s_1,s_2)\right) = 2\sigma^2 \left\{\sqrt{(t_1 - s_1)^2 + (t_2 - s_2)^2}\right\}^{2H}. \qquad (3)$$

Figure 4 demonstrates the values of the random field at two points. They are separated by distance d and denoted by A and B. In this notation, a fractional Brownian motion has the property that the difference between A and B, A-B, is Gaussian with mean 0 and variance $2\sigma^2 d^{2H}$. That is,

$$\mathrm{Var}(A - B) = E\left\{(A - B)^2\right\} = 2\sigma^2 d^{2H}. \qquad (4)$$

The mid-point displacement algorithm begins by finding the corner values A, B, G, and D of a two dimensional array (figure 5) and then, at each stage, finds values at other positions relative to these in the following steps:

1) interior
2) edge
3) interior
4) interior (symmetric).

The points in step four above are symmetric about the diagonal to those in step three.

21
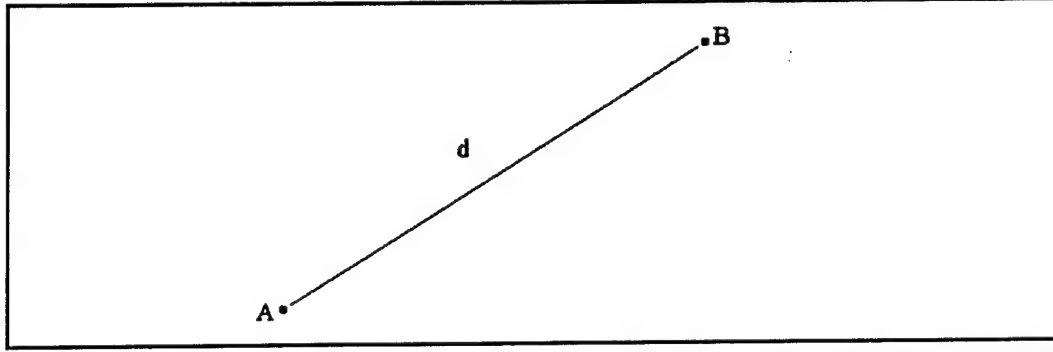
Figure 4. A picture of the relative positions, A and B, within the random field, separated by distance d.
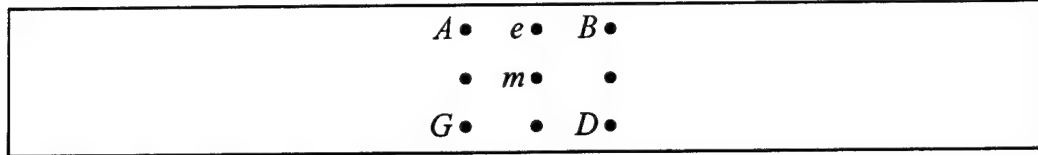
Figure 5. Graphical aid to assist in defining the detailed steps of the mid-point displacement algorithm.

The initial step of this algorithm obtains values for the four corners A, B, G, and D by independently sampling a Gaussian distribution with zero mean and variance $\sigma^2 d^{2H}$ (figure 5). It is easy to show that equation (4) is satisfied for any two adjacent values on an edge. For example, A and B:

$$\mathrm{Var}(A - B) = \mathrm{E}\left\{(A - B)^2\right\} = \mathrm{E}\left\{A^2\right\} + 2\,\mathrm{E}\left\{AB\right\} + \mathrm{E}\left\{B^2\right\} \tag{5}$$

Using the independence of A and B we have,

$$\mathrm{Var}(A - B) = \mathrm{E}\left\{A^2\right\} + \mathrm{E}\left\{B^2\right\} = \sigma^2 d^{2H} + \sigma^2 d^{2H} = 2\sigma^2 d^{2H}. \tag{6}$$

This verifies equation (4) for the pair of points A and B. The same result obtains for pairs (B,D), (A,G), and (G,D).

Equation (4) is not satisfied for the diagonal pairs (A,D) and (B,G). From the independence of A and D and the definition of A and D we have, just as above,

$$\text{Var}(A - D) = 2\sigma^2 d^{2H} \quad . \tag{7}$$

But if equation (4) were to be satisfied by the pair A and D, which is separated by a distance of $d\sqrt{2}$, we would have the following different value for $\text{Var}(A - D)$:

$$\text{Var}(A - D) = 2\sigma^2 \left(d\sqrt{2}\right)^{2H} \quad . \tag{8}$$

The mid-point displacement algorithm uses two recursive steps. One is used to find the value of an interior center point m (figure 5). The other is used to find the value of an edge point, such as that labeled e. The labels found in figure 5 are used in the following discussion to explain the respective recursive steps.

In figure 5, the value m at the center of a square with corner values A, B, G, and D is determined by the following formula:

$$m = \frac{(A + B + G + D)}{4} + \varepsilon \tag{9}$$

where ε = an independent Gaussian variate with zero mean and variance

$$\text{Var}(\varepsilon) = \sigma^2 \left(\frac{d}{\sqrt{2}}\right)^{2H} \quad . \tag{10}$$

In this step, the variance of the independent variate decreases by a factor of $(1/2)^H$. With this assignment for the variance, equation (4) will not be satisfied for the pair (A,m) even at stage one of the algorithm.

In figure 5, the value e at an edge is found by:

$$e = \frac{(A + B + m)}{3} + \eta \tag{11}$$

where η = an independent Gaussian variate with zero mean and variance

$$\mathrm{Var}(\eta) = \sigma^2 \left(\frac{d}{2}\right)^{2H}.$$ (12)

Equation (4), therefore, will not be satisfied for the pairs (e,A) and (e,B).

The process for the first two stages is illustrated by figures 6 and 7. Figure 6 illustrates the points determined by stage one. The center point is labeled one and the edge points, labeled two through five, are determined in this stage. No interior points are determined in parts three and four of stage one.

			interior: 1
A	4	B	edge: 2,3,4,5
2	1	3	interior: none
G	5	D	interior: none

Figure 6. Depiction of stage one of the mid-point displacement algorithm. The values at locations one through five are chosen.

Figure 7 shows the result of stage two. In part one of stage two the interior points six, seven, eight, and nine are found in that order. In part two, the edges 10 through 17 are found. In part three, interior points 18 and 19 are found. Finally, points 20 and 21 are determined.

A	12	4	16	B	interior: 6,7,8,9
10	6	20	8	11	edge: 10-17
2	18	1	19	3	interior: 18,19
14	7	21	9	15	interior: 20,21
G	13	5	17	D	

Figure 7. Depiction of stage two of the mid-point displacement algorithm. The values at locations six through 21 are chosen.

Figure 8 shows an example of a texture map produced by this algorithm. The mid-point displacement algorithm produces an image that only approximates a fractional Brownian motion. As the algorithm produces finer grids, the variance of the difference of two nearby points roughly gets smaller as prescribed by equation (4). Appendix A describes another algorithm, based on

equation (4). Appendix A describes another algorithm, based on this concept, [9] which does a better job. The other algorithm was not discovered until this project was well under way; otherwise, it would have been afforded a more prominent place in this report.
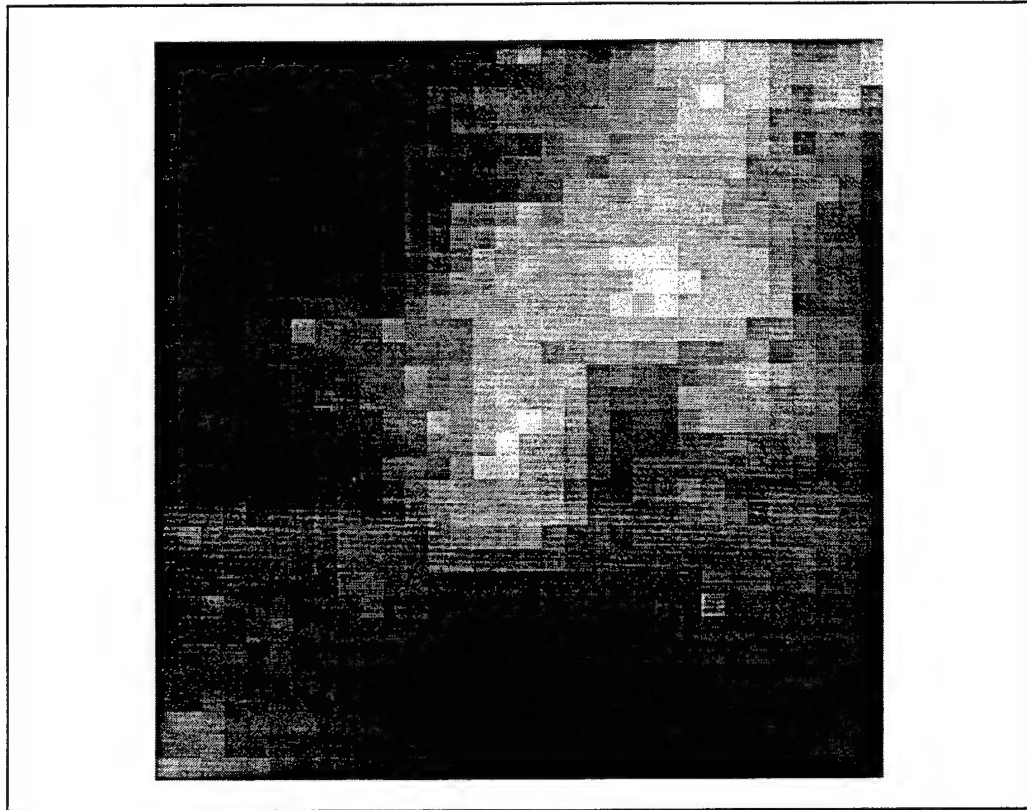
Figure 8a. The texture image output by the mid-point displacement algorithm.
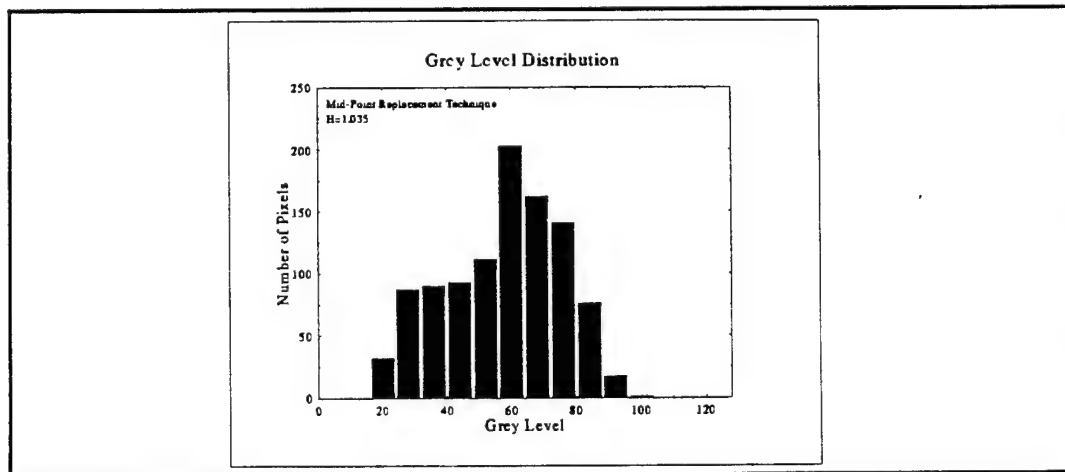
Figure 8b. The histogram of grey levels for the texture image of figure 8a.

3.2   Grey-Level Co-Occurrence Matrix Algorithm

The Grey-Level Co-Occurrence Matrix Algorithm (GLCM) method for building textures presented here is based on the algorithm described in the paper by G. Lohmann. [11] The GLCM has been often used as a basis for the calculation of secondary features such as contrast, correlation, entropy, and homogeneity, which are used to categorize and segment textural features in images. In this algorithm, a set of four GLCMs is used to provide feature vectors. Lohmann reports that the four primary GLCMs (horizontal, vertical, left- and right-diagonal) contain sufficient information to synthesize textures that very closely resemble textures from remotely sensed images of the Landsat/TM and ERS-1/AMI sensors. The idea is to determine how this algorithm performs on our type of imagery.

Before discussing the algorithm we present some basic definitions. Consider pairs of pixels separated by distance d at some angle ϕ. Generally, distances of one pixel and angles of 0, 45, 90, and 135° are used. The $(d = 1, \phi = 0°)$-pairs are horizontally adjacent, the $(d = 1, \phi = 90°)$-pairs are vertically adjacent, the $(d = 1, \phi = 45°)$-pairs are right-diagonal neighbors, and the $(d = 1, \phi = 135°)$-pairs are left-diagonal neighbors. If n denotes the number of grey levels in the image, then the (d, ϕ)-co-occurrence matrix C_ϕ is an n by n matrix, where an entry (i,j) of C_ϕ denotes the number of pairs of pixels separated by distance d at angle ϕ, which have grey values i and j.

The following examples of 0, 45, 90, and 135° co-occurrence matrices (equations (14) through (17)) for the image called "target" below. Equation (13) illustrate the above definitions:

$$\text{target image} = \begin{bmatrix} 0 & 1 & 0 \\ 2 & 1 & 2 \\ 0 & 2 & 1 \end{bmatrix} \tag{13}$$

$$C_0 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 2 & 0 \end{bmatrix}, \tag{14}$$

26

$$C_{45} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \qquad\qquad (15)$$

$$C_{90} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 2 & 1 & 0 \end{bmatrix}, \text{ and} \qquad\qquad (16)$$

$$C_{135} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}. \qquad\qquad (17)$$

The C_ϕ matrices are 3 by 3 in size because there are 3 grey levels in the target image. If there were 16 grey levels in the image, then the GLCM would be 16 by 16, and so on.

The manner in which the grey level co-occurrence matrix is populated follows. Referring to figure 9, all pairs of pixels in the target image that are separated by a distance of 1 pixel at an angle of $0°$ are determined and the number inserted into the $0°$ GLCM, C_0. For example, the $0 \rightarrow 1$ situation occurs once; therefore, a 1 is placed in the GLCM at the position $(0,1)$ as shown in panel (a) of figure 10, while the $2 \rightarrow 1$ situation occurs twice resulting in a 2 at position $(2,1)$ as shown in panel (b) of figure 10. Continuing in this manner results in the C_0 given by equation (14).

$$0 \rightarrow 1 \rightarrow 0$$
$$2 \rightarrow 1 \rightarrow 2$$
$$0 \rightarrow 2 \rightarrow 1$$
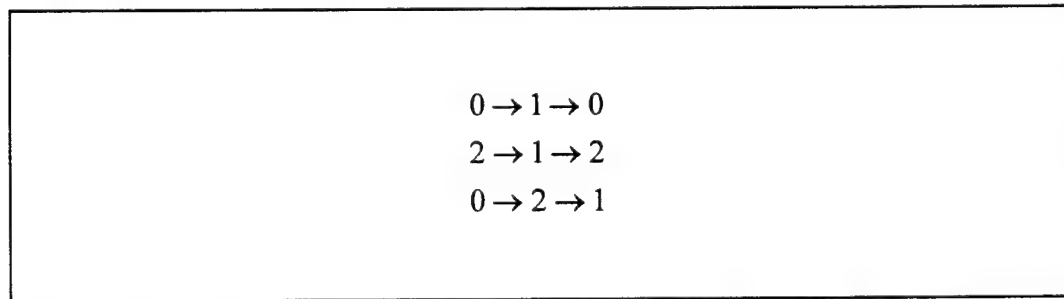
Figure 9. The pairing technique used to populate the 0° GLCM with a distance of 1 pixel separating pairs in the horizontal direction.
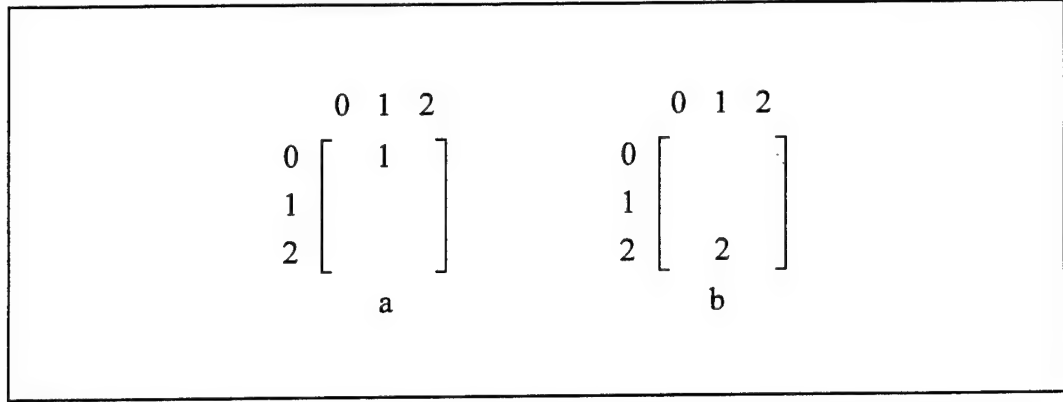
Figure 10. Filling the 0° GLCM of the target image. Panel (a) is the result of counting the number of (0→1) grey levels separated by a distance of 1 pixel in the horizontal direction (=1), while panel (b) is populated from the number of (2→1) occurrences (=2). The completed matrix is given in equation (14).
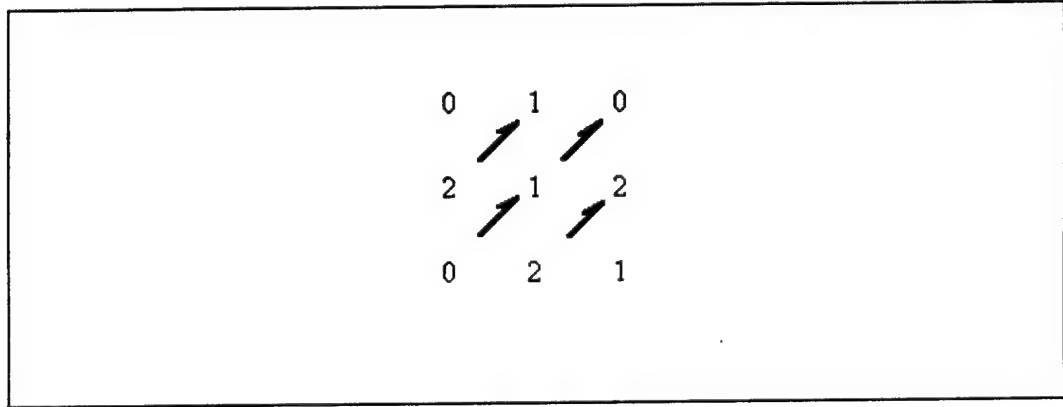
Figure 11. The pixel pairs having a 45° relationship and whose numbers are used to populate the C_{45} matrix.

Similarly, for the 45° GLCM, we count the number of occurrences of pixel pairs separated by one space in the 45° direction as pictured in figure 11. And, as with the 0° GLCM, the 45° and remaining GLCMs are given in equations (15) through (19).

In the initial step of the algorithm, a random image is generated that has the same grey level histogram as the target image whose texture we are trying to replicate. The histogram is calculated from the row sums of the 0, 45, and 90 degree GLCMs:

$$\text{hist}[i] = \text{rs0}[i] - \text{rs45}[i] + \text{rs90}[i] + \text{tr}[i] \tag{18}$$

28

where

$\text{hist}[i] = $ # of occurances of greylevel i in the "target" image

$\text{rs0}[i] = $ sum of entries of row i of the 0 degree GLCM

$\text{rs45}[i] = $ sum of entries of row i of the 45 degree GLCM

$\text{rs90}[i] = $ sum of entries of row i of the 90 degree GLCM.

The vector tr[i]is the top right corner occupancy vector. That is,

$$\text{tr}[i] = \begin{cases} 1, & \text{if greylevel } i \text{ occurs at the bottom right corner of "target"} \\ 0, & \text{otherwise} \end{cases}$$

The values of these vectors, for our example, are given by the following equations:

$$\text{rs0} = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}, \tag{19}$$

$$\text{rs45} = \begin{bmatrix} 1 \\ .1 \\ 2 \end{bmatrix}, \tag{20}$$

$$\text{rs90} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \tag{21}$$

$$\text{rs135} = \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix}, \text{ and} \tag{22}$$

$$\text{tr} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}. \tag{23}$$

Equation (23) can be verified in the example given by equation (13). To find the number of 2's in the target image we calculate:

$$
\begin{aligned}
\text{hist}[2] &= \text{rs0}[2] - \text{rs45}[2] + \text{rs90}[2] + \text{tr}[2] \\
&= 2 - 2 + 3 + 0 \\
&= 3.
\end{aligned}
\tag{24}
$$

Note that the indices for the GLCM matrices begin with 0. The formula in the computer code (appendix C) for the histogram is slightly different from equation (23) in order to reflect the convention of placing the origin of an image at the top left corner rather than at the bottom left as we have here.

This histogram is used to randomly populate a matrix to produce an initial image. Equation (28) is an initial image with the same histogram as the example given by equation (13) which is repeated here as equation (33). Equations (29) through (32) and (34) through (37) are the four GLCMs associated with the two images. The sum of the absolute differences of corresponding positions of the GLCMs is a measure of the closeness of the sets of the GLCMs. This measure of differences, known as the Manhattan metric, is calculated for the individual GLCM matrices and summed to 19 in equation (38). It should be noted with the Manhattan metric there are a variety of distance measures used to describe the separation of two vectors. Three of these are the Euclidean Distance, the Manhattan (or City Block) Distance, and the Square Distance. Mathematically, these are, respectively:

$$
E = \sqrt{\left[\sum (X_i - Y_i)^2 \right]}
\tag{25}
$$

$$
M = \sum |X_i - Y_i|
\tag{26}
$$

$$
S = \text{MAX} \, |X_i - Y_i| \, .
\tag{27}
$$

In the case of the Manhattan Distance, if binary vectors replace the vectors X_i and Y_i, then the distance is known as the Hamming Distance. [13]

$$\text{initial} = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 0 & 0 \\ 0 & 2 & 2 \end{bmatrix}, \tag{28}$$

$$C_0 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \tag{29}$$

$$C_{45} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \tag{30}$$

$$C_{90} = \begin{bmatrix} 0 & 2 & 1 \\ 0 & 1 & 0 \\ 2 & 0 & 0 \end{bmatrix}, \tag{31}$$

$$C_{135} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, \tag{32}$$

$$\text{target} = \begin{bmatrix} 0 & 1 & 0 \\ 2 & 1 & 2 \\ 0 & 2 & 1 \end{bmatrix}, \tag{33}$$

$$C_0 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 2 & 0 \end{bmatrix}, \tag{34}$$

$$C_{45} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \tag{35}$$

$$C_{90} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 2 & 1 & 0 \end{bmatrix}, \tag{36}$$

$$C_{135} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{37}$$

$$\begin{aligned} \Delta &= \delta C_0 + \delta C_{45} + \delta C_{90} + \delta C_{135} \\ &= 4 + 6 + 3 + 6 \\ &= 19. \end{aligned} \tag{38}$$

We begin a sequence of pairwise exchanges in the initial image in an attempt to arrive at an image that is closer to the original target image as measured by the distance metric. If the distance between the new image and the target image is less than the distance before the pairwise exchange, we make the exchange that results in a new image in the iterative process. However, if there is no improvement, we may make the swap of the two grey level values anyway. We can call this a "wildcard" switch. This is done with probability p where:

$$p = \frac{1}{1 + \exp\left(\frac{\Delta}{T}\right)} . \tag{39}$$

Here,

Δ = the current error

and

T = a temperature

that is slowly cooled down. This method is a way of simulating an annealing process (see appendix B). The purpose of this wildcard switch is to allow exit from a "local minimum."

The algorithm halts if the number of successful attempts at switching falls below some predetermined percentage of the number of pixels in the target image lattice. This percentage is set to one percent or less of M, the number of pixels in the target image. After each iteration of M attempts at switching, the procedure either halts or the iteration continues with a new cooler

annealing temperature T. As T gets smaller, the probability p also gets smaller and eventually there are fewer and fewer "wildcard" switches.

When two grey levels are swapped in an image, very few (eight, at most) of the entries of a co-occurrence matrix are affected. The entire co-occurrence matrix does not have to be recomputed. To illustrate the effect on the $0°$ GLCM C_0 by way of example, suppose we swap the two grey levels that are noted by an asterisk in figure 12. Then, figure 13 shows the contribution of these grey levels to the GLCM, and figure 14 illustrates the changes that occur after the swap has occurred. The contributions to the C_0 GLCM by the new ordered pairs (1,0), (0,1), (1,2), (2,0) are shown in the right-hand side of figure 14. The new C_0 GLCM is obtained by subtracting the first change matrix from the old C_0 GLCM and then adding the second change matrix, as illustrated in figure 15. The four locations in the new C_0 GLCM, which have changed, are flagged by an asterisk. The two locations flagged by a double asterisk could have changed, but did not because of cancellations. In the most general case, a maximum of eight locations in a GLCM could change as the result of swapping the two grey levels.

It is apparent from this example that it is important to keep track of grey levels that are at the tail and tip of the arrows in figures 13 and 14. The notation that is used in the computer code *glcmtex.c* (appendix G) to label these points is shown in figure 16, which is an annotated version of figure 11. The P- denotes the "predecessor" of position P, the grey level at position P- is the row index to the C_0 GLCM entry contributed by the pair 1→2. Similarly, P+ indicates the successor of position P. The grey level of 1 at location P+ is the column index of the C_0 GLCM contribution made by the pair 2→1.

In terms of these labels, the computer code *glcmtex.c* considers three cases in updating the GLCM resulting from a swap:

> case 1 - point Q coincides with point P-;
> case 2 - point Q coincides with point P+;
> case 3 - all of the rest.

These labeling conventions are used in the functions *change* and *update_glcm* in *glcmtex.c* and are explained here for better understanding of the code. The computer code also checks to see whether points P-, P+, Q-, and Q+ fall

outside of the image. For example, if point P is at a left edge, the point P- does not exist when considering the case of a 0° GLCM.

Figure 17 is an example of a texture map produced by the GLCM-based algorithm. The textural properties for this map are listed in table 2.

$$\text{initial} = \begin{bmatrix} 1 & 2* & 1 \\ 1 & 0* & 0 \\ 0 & 2 & 2 \end{bmatrix},$$

Figure 12. This figure, first in a series (12 through 15), demonstrates the affect the swapping of two grey levels has on the GLCM. In this case, the *-ed values are the two positions being swapped.

$$\begin{matrix} 1 & \to & 2 & \to & 1 \\ 1 & \to & 0 & \to & 0 & \text{yields:} & \begin{bmatrix} +1 & & \\ +1 & & +1 \\ & +1 & \end{bmatrix} \\ 0 & & 2 & & 2 \end{matrix}$$

Figure 13. The second figure, of the series (12 through 15), indicates the portion of the 0° GLCM that is affected by the values being swapped, and by how much it is affected.

34

$$
\begin{array}{ccccc}
1 & \rightarrow & 0 & \rightarrow & 1 \\
\\
1 & \rightarrow & 2 & \rightarrow & 0 \qquad \text{yields:} \qquad \begin{bmatrix} & +1 & \\ +1 & & +1 \\ +1 & & \end{bmatrix} \\
\\
0 & & 2 & & 2
\end{array}
$$

Figure 14. The third figure, of the series (12 through 15), indicates the portion of the 0° GLCM affected after the swap, and by how much.

$$
\begin{bmatrix} 0* & 1* & 1 \\ 1** & 0 & 1** \\ 1* & 0* & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}
$$

new C_0 GLCM   old C_0 GLCM   1st change   2nd change
                                        matrix   matrix

Figure 15. The last figure, of the series (12 through 15), demonstrates the technique used to determine the new GLCM without completely recalculating the whole GLCM.

35

$$\begin{bmatrix} \begin{matrix} \text{P-} \\ 1 \end{matrix} \rightarrow \begin{matrix} \text{P} \\ 2 \end{matrix} \rightarrow \begin{matrix} \text{P+} \\ 1 \end{matrix} \\ \\ \begin{matrix} \text{Q-} \\ 1 \end{matrix} \rightarrow \begin{matrix} \text{Q} \\ 0 \end{matrix} \rightarrow \begin{matrix} \text{Q+} \\ 0 \end{matrix} \\ \\ 0 \qquad 2 \qquad 2 \end{bmatrix}$$

Figure 16. The labeling of the various elements of the GLCM, in this case the C_0 GLCM, to keep track of how the swaps are made.
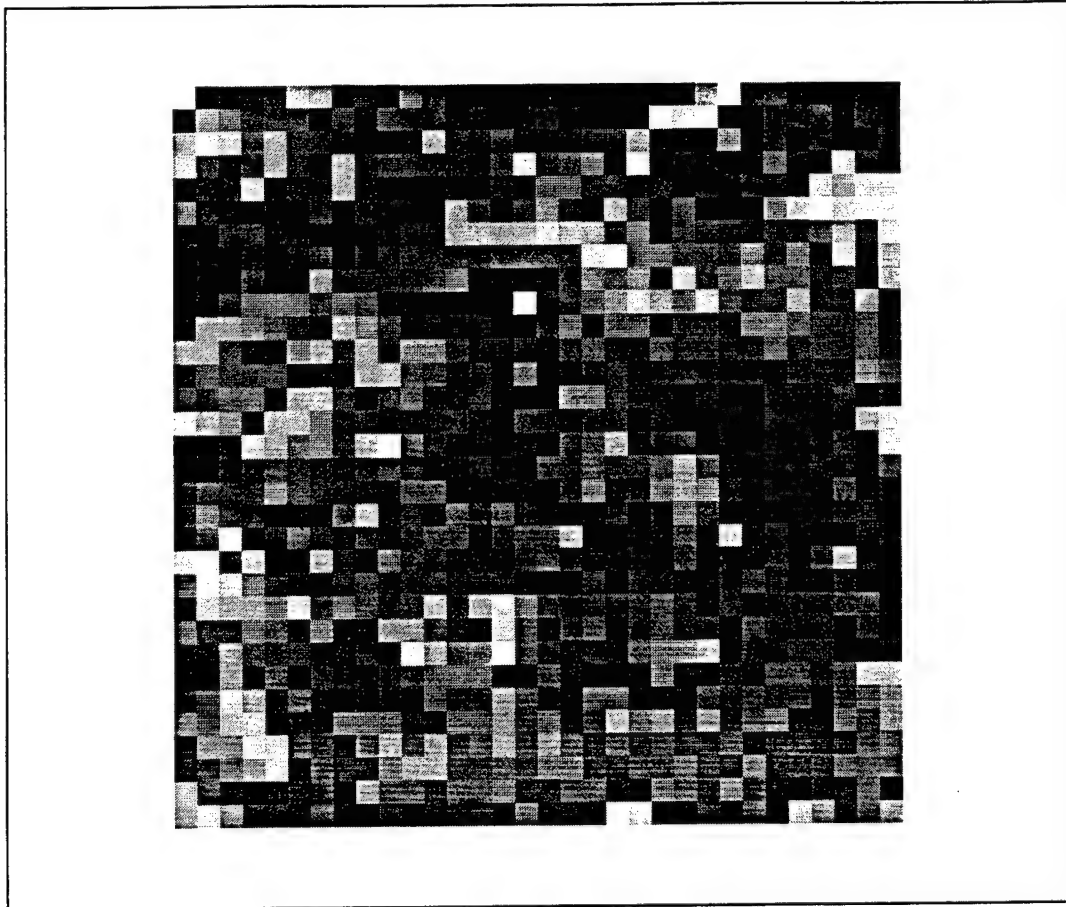
Figure 17a. A texture map created through the use of the GLCM-based technique.
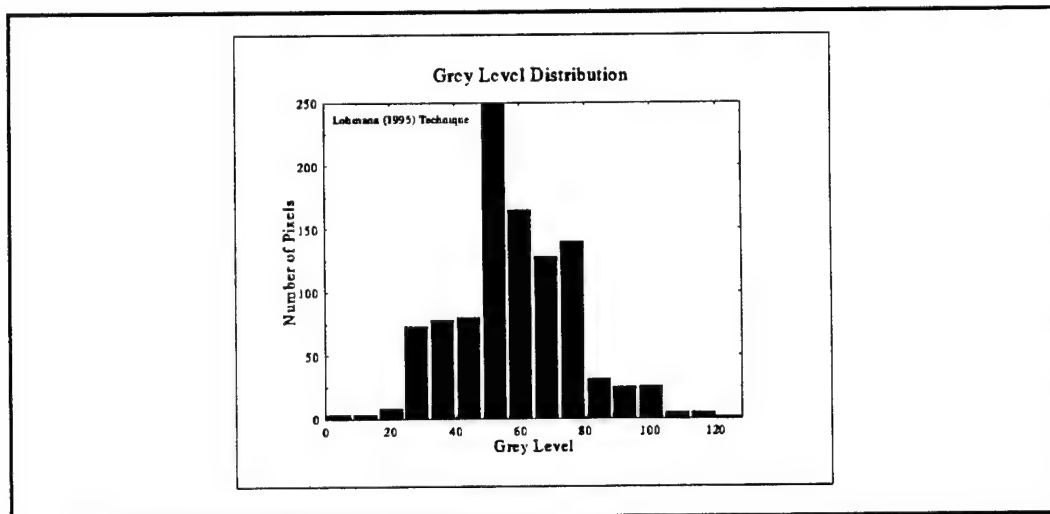
Figure 17b. The histogram of grey levels for the GLCM-based texture map of figure 17a.

3.3   Correlation Length Algorithm (Current Texture Generation Process)

The two-dimensional AR model used in this study is found in the report by Sabol and Balick. [14] The texture generation code is listed there. The PASCAL code was translated into C for this study, and is included in appendix D.

In paragraph 6.4 of appendix G of the *SWOE Final Report*, [2] the basic methodology is laid out:

> An empirical approach was taken because of the lack of general theory or models on thermal IR texture. Using this approach "homogeneous" surfaces (grass, bare soil, trees, tree lines, etc.) of interest were imaged for the times (or under similar conditions), the synthetic scenes were generated. Textural features of these surfaces were measured and input to an AR texture generator program, which generated isotropic Gaussian texture maps. These maps were used by the rendering software to texture the polygons.

> Numerous approaches to texturing were reviewed and a simplified two-dimensional AR model was selected. It uses correlation length in vertical and horizontal directions and brightness mean and standard deviation as input parameters for a kernel (process of order [1,1]). The model was developed from an AR model used in the US Air Force Infrared Modeling and Analysis (IRMA) image modeling system.

Study of the computer code reveals that texture is generated one pixel at a time, using the following equation:

$$P[i,j] = a1 * P[i,j-1] + a2 * P[i-1,j] - a1 * a2 * P[i-1,j-1] + R[i,j] \qquad (40)$$

where :

$i =$ current low location

$j =$ current column location

$P[i, j] =$ pixel at location $[i,j]$

$R[i, j] =$ random number generated at location $[i, j]$.

The random number, $R[i, j]$, is constrained such that following conditions are met:

$$\text{mean} = m(1 - a1 - a2 + a1 * a2)$$
$$\text{variance} = s^2 \left[1 - a2 * a2 - a1 * a1(a2 * a1)^2\right]$$

where:

m = mean gray level of the image

s = standard deviation of gray level in image

$a1 = \exp(1/h)$

$a2 = \text{ext}(1/v)$

h = horizontal correlation length of gray level variation in image

v = vertical correlation length of gray level variation in image.

Note that equation (40) is not the same as the corresponding equation on page five of the Sabol and Balick. [14] Equation (40) as shown here reports what is in the PASCAL code.

Sabol and Balick cite that several modifications were made to the original AR model. Modifications are listed, with our comments:

1) Texture used in this application is isotropic therefore h and v are set to the same value.

2) Mean and standard deviation are fixed at 128 and 32, respectively.

39

(In our study, the mean and standard deviation of a target-texture image are used as input parameters to the computer program to fix the mean and standard deviation of the generated image. The model automatically produces an image with the specified mean and standard deviation.)

3) Because the AR model is only a first-order model (based on immediate neighbors), artifacts (vertical and horizontal striations) are generated for long correlation lengths (such as, correlation length >> 1). A moving-average filter (low-pass filter) is applied to the output of the AR module to reduce these artifacts. Only the diagonal elements are included in the filters, and all are given equal weight. The size of the filter is set to the correlation length.

(In our study of the PASCAL (procedure *moving-average)*, we noted that the low-pass filtering was accomplished by a convolution filter of fixed 3 by 3 size.)

4) The low-pass filtering described above compresses the distribution of values in the texture map. Larger moving average filters result in greater compression of the distribution. The desired mean of 128 and standard deviation of 32 for the distribution is restored using a histogram specification algorithm, which does a one-to-one remapping of digital values to restore the desired standard deviation to 32.

(For our study, the remapping for the filtered image was not performed.)

5) Internally, a 300 by 300 texture array is generated, but only the center 256 by 256 is output. This eliminates initialization effects and provides a large enough area to apply the moving average filter.

(For our study a 32 by 32 pixel image was generated. No attempt was made to generate a larger image from which to extract a 32 by 32 subimage was made.)

In the PASCAL listing for the function RAN1, in the Sabol and Balick report, the constant M2 is defined M2 = 124456. [14] This varies from Press et al.

where the function RAN1 is listed with a PARAMETER M2 = 134456. [15] There is a single digit discrepancy between these two numbers. According to Press, the choice of constants used to implement a random number generator (of which this parameter, M2, is one) is critical to the proper operation of the generator. However, it is beyond the scope of this report to delve into this aspect further.

Figure 18 shows an example of both the raw and smoothed texture map produced by the algorithm. The textural properties of these maps are given in table 2.

(a)

(b)
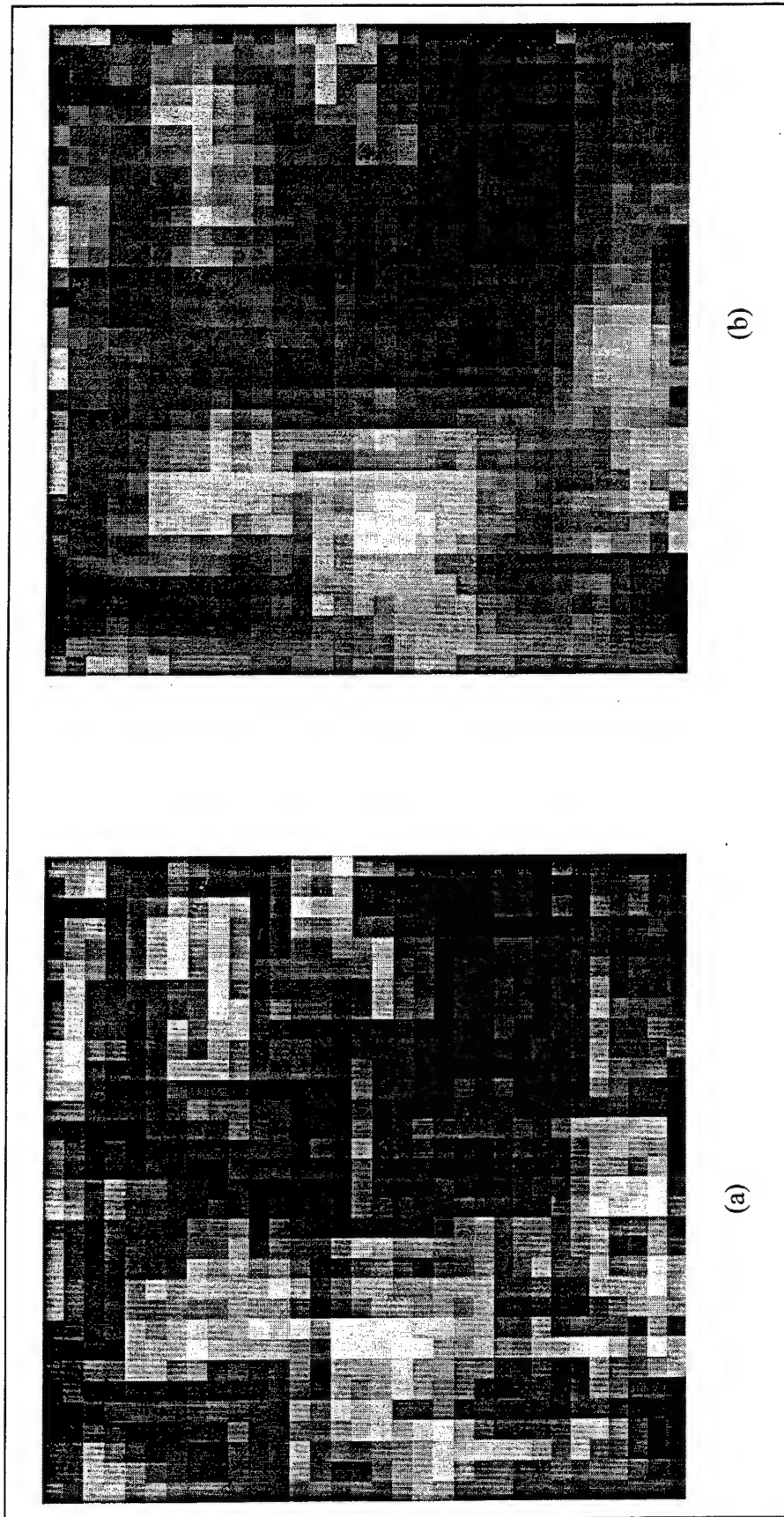
Figure 18a. A typical raw image (a) and the resultant filtered image (b), which are created from the correlation length algorithm.
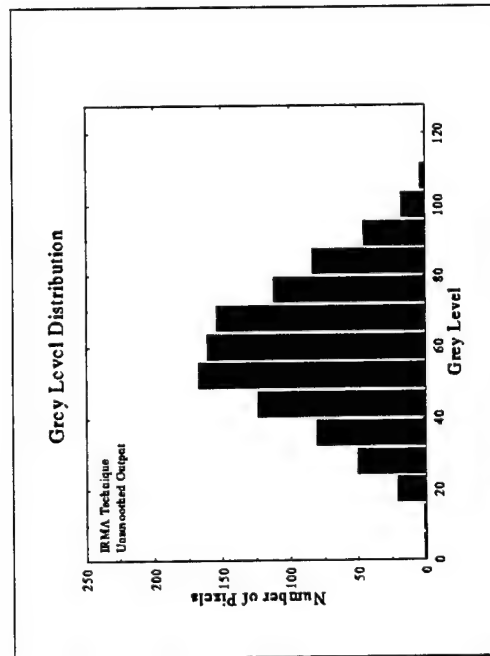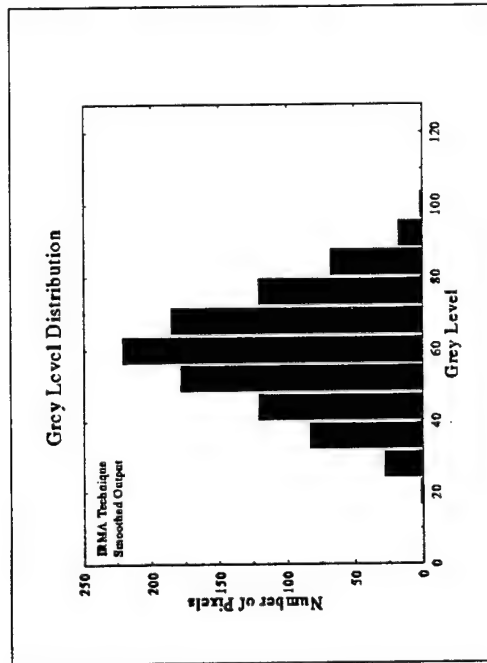
Figure 18 (b). Panel (c) is the grey level histogram for the image in panel (a). Panel (d) is the grey level histogram for the image in panel (b).

4. Comparisons and Discussion

The target texture, which is the one to be duplicated by the various algorithms, is shown in figure 19 (a). This map is a 32 by 32 image (segmented from the image/texture map of figure 1) that has texture properties given in table 1. For the correlation length technique, we present a typical raw image (unsmoothed) in figure 19 (b). In figure 19 (c) the midpoint displacement result is shown, and in figure 19 (d), the grey level co-occurrence matrix technique output is demonstrated. Table 2 contains the results of applying the various texture metrics to these texture maps. The table headings are acronyms for the metrics discussed in section 2. With the exception of the mean and variance, these metrics measure the second order statistics of the images. All have been used in many studies of the textural properties of images. The target texture is shown in figure 3, the texture generated by the mid-point replacement algorithm is shown in figure 8, figure 17 shows the GLCM-generated texture, and the IRMA (raw and smoothed) is shown in figure 18. Figure 19 repeats many of these maps on smaller scale for side-by-side comparison. Reference should be made to the text for definition of the table headings. More details on the form and application of these metrics can be found in Bleiweiss. [3] *MEAN* is the average grey level of the texture map (ranges from 0 to 128), *VAR* is the variance of the grey levels in the texture map, *FRAC D* is the fractal dimension, *ACL* is the autocorrelation length, *CONTR*, *CORR*, *ENTRPY*, and *HOMOG*, are GLCM metrics called "contrast", "correlation", "entropy", and "homogeneity," respectively. The GLCM metrics are somewhat self-explanatory in meaning.

Table 2. Texture measure values for generated textures shown as examples in various figures

SOURCE	MEAN	VAR	FRAC D	ACL	CONTR	CORR	ENTROPY	HOMOG
Target Texture	57.247	338.116	1.965	3.537	222.219	0.667	4.644	0.014
Mid-pt replace	56.740	338.052	1.774	9.312	22.554	0.967	6.641	0.002
GLCM	57.247	338.115	1.804	1.410	376.695	0.427	4.776	0.013
IRMA	58.790	338.043	2.117	3.539	164.708	0.760	7.180	0.001
IRMA(Smoothed)	58.885	211.740	1.593	4.314	54.050	0.873	6.799	0.001

Table 3. Texture measure values for generated textures

Source		MEAN	VAR	FRACD	ACL	CONTR	CORR	ENTRPY	HOMOG
Target Texture		57.247	338.116	1.965	3.537	222.219	0.0667	4.544	0.014
Mid-pt replace	average	56.756	336.775	1.920	11.116	14.393	0.978	6.419	0.002
	std.dev	0.034	4.268	0.139	8.274	8.111	0.012	0.197	0.00052
GLCM	average	57.247	338.116	2.163	1.452	394.532	0.399	4.802	0.013
	std.dev	0.000	0.000	0.134	0.402	20.833	0.032	0.015	0.00020
IRMA	average	56.027	327.618	2.159	3.055	160.203	0.748	7.126	0.001
	std.dev	5.013	67.931	0.156	0.811	13.776	0.045	0.051	0.00010
IRMA	average	56.022	213.424	1.958	3.813	53.767	0.864	6.760	0.001
(smoothed)	std.dev	5.059	63.775	0.189	1.211	5.503	0.031	0.101	0.00017

Each algorithm was run 101 times, each with a different pick from the random number generator. Each algorithm uses a random number generator; however, the specific use is somewhat different from algorithm to algorithm. The results of these calculations are shown in table 3, which are the same as table 2 except the tabulated values applying to a single map are the average and standard deviation of the results from application of the metrics to the 101 maps. The sources labeled IRMA and IRMA (SMOOTHED) are autocorrelation length algorithm output. The variation is due to the variation introduced by the use of random number generators. The histograms from which these averages and standard deviations were determined are shown in appendix D.

Reference to table 3, tells the story about the relative ability of the various algorithms to replicate the target texture as measured by the various metrics used in this study:

- mid-point replacement algorithm - The texture map created by this technique is similar to the target texture as measured by the fractal dimension (though not exact) but is different, as measured by the other metrics.

- autocorrelation length (IRMA) algorithm - This texture map (unsmoothed) is close as measured by the autocorrelation length, but is quite different as measured by the other metrics (with the possible exception of the fractal dimension which is

similar to that of the target). The smoothed output presents an autocorrelation length much closer to the target. The fractal dimension is nearly the same as for the target, in fact much closer than that of the mid-point replacement algorithm. The GLCM metrics all become worse under the action of the smoothing function.

- GLCM algorithm - The texture map created by this algorithm is about the same as the autocorrelation length algorithm as measured by the fractal dimension and, except for the entropy measure (which shows good agreement with the target entropy) all of the other GLCM metrics show that this map is different from the target map.

It would appear, then, that the autocorrelation-based technique (at least the smoothed version) is best able to create an image whose autocorrelation is nearly the same for target and synthetic, while the other techniques seem to not be able to replicate themselves very well. A possible exception is the IRMA and GLCM ability to yield a fractal dimension close to that of the target. It remains to do a sensitivity study to determine the discrimination ability of these various metrics as their values change by small amounts. For example, is there much difference between a texture whose fractal dimension is 2.75 and one whose value is 2.85, and so on. Visual inspection may favor the GLCM algorithm, though this may be due to the level of pixellation in the two maps and not truly comparable texture.

Why then, do the synthetic SWOE images have textural metrics, especially the autocorrelation length, so different from the "real" images? [12] More than likely, the answer lies in the way the texture was measured for input to the SWOE model, the way in which the texture was applied to the synthetic images, and the way in which the texture was measured in the real and synthetic images. For example, the scene that was imaged for textural characteristics was seen at a different perspective view, and with a different spatial resolution than that of the real images and, the application of the synthetic texture is made to the scene elements as seen from the imager's perspective (but with a texture as seen from a different perspective) and not to the scene elements before they are rendered into an image.

In fact, because the properties that create texture do not yield imagery that is invariant to the viewpoint, it is not at all clear that using such a simple approach would work at all. The only exception is a simple approach where texture is extracted from the real images and then pasted into the synthetic images. This really becomes a hybrid technique, not a true synthetic technique. For an invariant approach, the texture should be applied as a variation of small scale relief and optical properties (assuming that the thermal properties are homogeneous at this scale) to the large scale relief. Then, when the scene is rendered into an image, the texture is properly mapped.

Future work may also look toward methods to parameterize texture based on some sort of environmental parameter. For example, one approach still with many limitations, would be to parameterize the measure of texture for a particular imager view and then apply the correct texture based on the parameterization. An example of how this would work is suggested by figure 20, a plot of a particular texture metric with time of day for the Yuma I field exercise of the SWOE JT&E. During that effort, the meteorological influences were quite even (day to day) so that diurnal trends in some of the textural metrics could be seen. During the Grayling field exercises, the meteorological influences were so variable that this simple correlation was lost to ready viewing; though it may possibly be extracted through some sort of multivariate analysis.
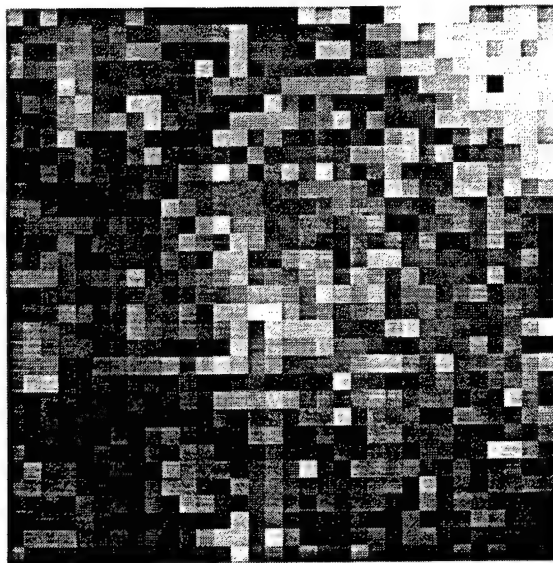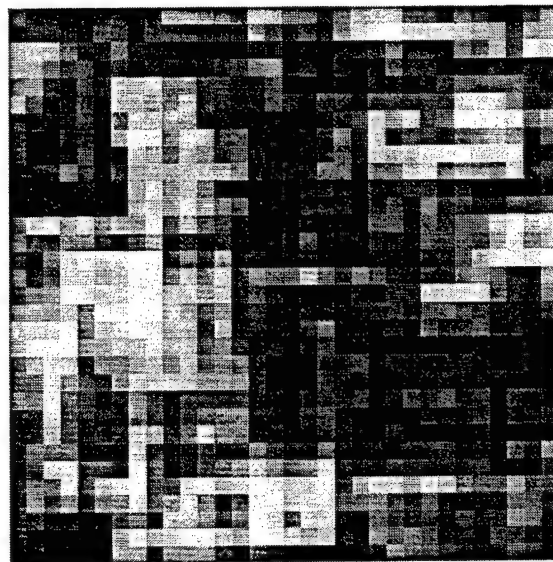
(a) Target (b) Correlation Length
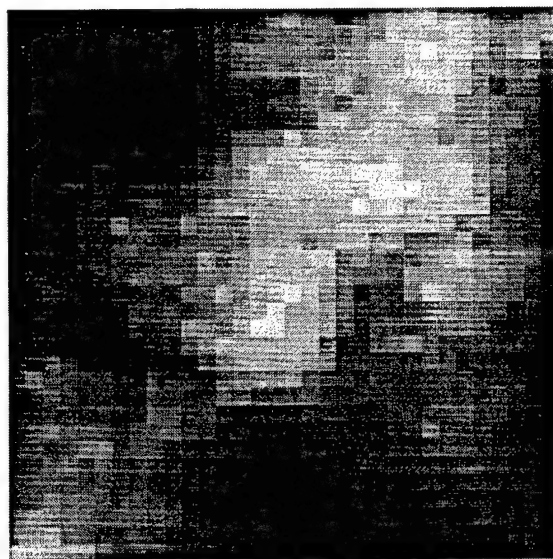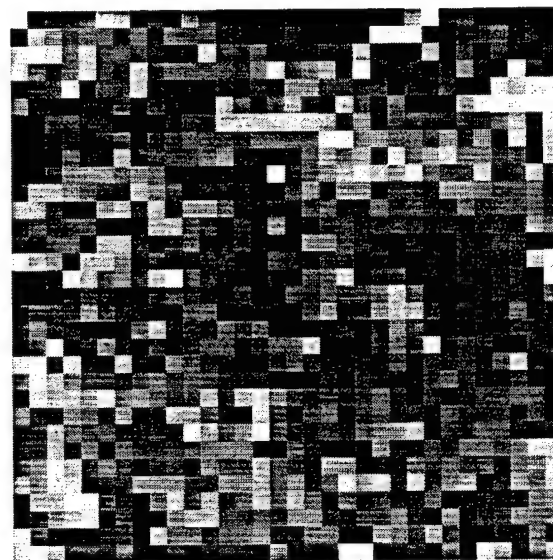
(c) Mid-Point Replacement (d) GLCM

Figure 19a. Four textures displayed in a common frame for comparison.
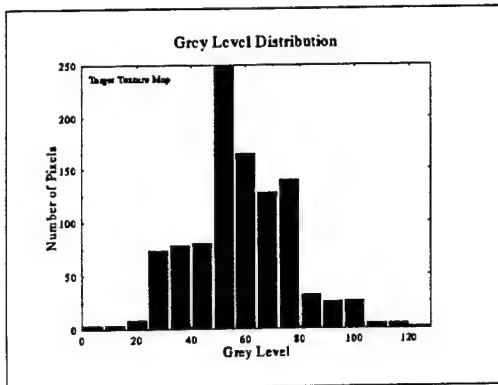
(e) Target
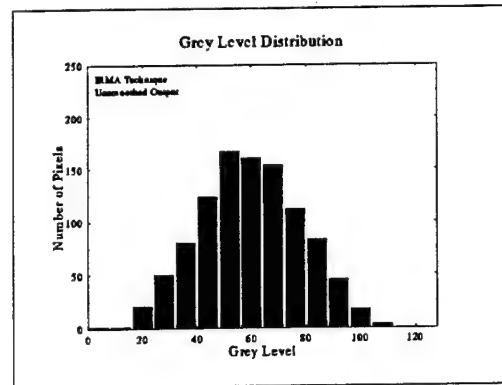
(f) Correlation Length
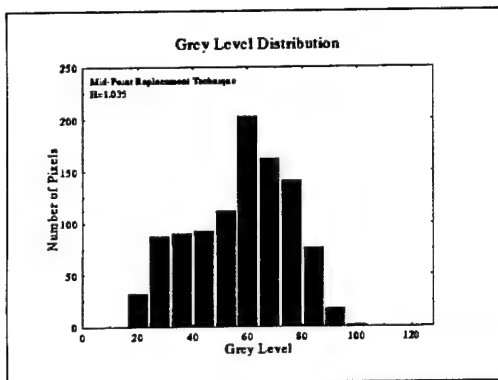
(g) Mid-Point Replacement
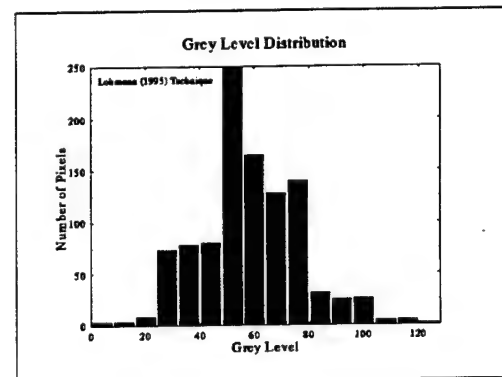
(h) GLCM

Figure 19 (b). The histograms for the four textures of panels displayed in a common frame, for a better visual comparison among them.

50
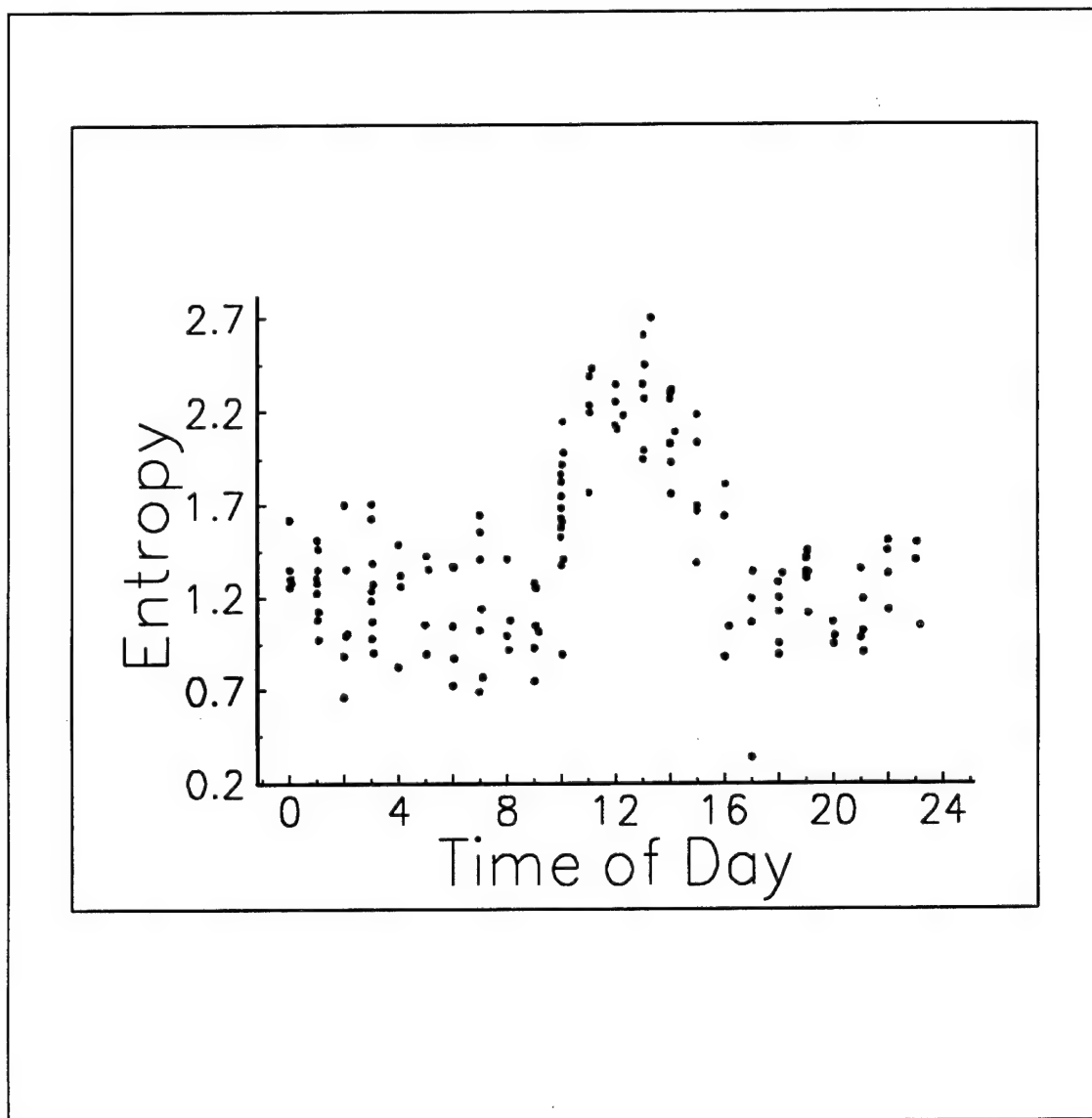
Figure 20. Plot of the entropy, a GLCM metric, versus time of day to demonstrate the diurnal trend experienced by a metric such as this, which could be used in a parameterization of a seed to produce synthetic texture.

5. Summary

We have demonstrated the generation of texture using three different algorithms in an attempt to compare their various abilities to replicate a given texture. The results of that demonstration are mixed, and though the autocorrelation technique was certainly no worse than the rest of the algorithms, it may be best suited for the task at hand. We feel that the synthetic imagery (as judged by the metrics) generated in the SWOE process did not compare more favorably with real image as to the textural properties, because of the way the texture was measured and then applied to the imagery. The dependence of texture on view angle was not taken into account. Also, the texture, as seen by real imagers (for the case studied in the SWOE effort) is not isotropic nor is it necessarily homogeneous over those scene elements thought to be homogeneous. Definitive answers remain until a more in depth study can be performed. The first step has been taken; the generation of the software and methodology to perform the study.

6. Future Directions

This effort has focused on possible quick solutions to the problem of simulating the real texture of a scene, in lieu of modeling the scene at very high-spatial resolution. We have shown that there does not appear to be a quick solution to the problem among the techniques explored here. The reasons for this include:

- texture generating algorithms are not good at producing a texture with the same measure as the seed;
- texture metrics are not good at measuring the texture;
- some of the texture metrics are misused. The assumptions underlying the model being used are not being met.

There may be a solution of the type we are looking for that will solve the problem. However, our research provided no solutions. Additional work might proceed in the following manner:

- further inventory of available texture metrics and their appropriate algorithms for use in generating texture;

- grade the ability of these metrics to discriminate against the type of textural features that we are likely to encounter.

Alternatives range from the very simple to the very sophisticated. For example, choose a limited variety of features that are likely to generate different textures, model them in detail, and then view them under a limited number of conditions:

- sparse grass at 10 levels of sparseness;
- bare dirt at different levels of roughness;
- and so on.

These could then be modeled under a limited number of seasons, diurnal conditions, and look angles to arrive at a library of textural features that could be selected to fill in the appropriate polygons of the scene.

To help in this modeling effort, we need to acquire a library of features that are to be modeled. This could include synthetic aperture radar (SAR) data to provide the relief component that goes into the model, field ground-truth measurements to ascertain the degree of homogeneity/inhomogeneity in the thermal and optical properties of the scene. Hyperspectral imaging might be useful here.

The Rochester Institute of Technology [16] as well as many other organizations, such as Purdue University [17] are conducting similar efforts. The Department of Defense and government directors of research such as the Army Research Office, may also provide pointers. A library of digital texture maps is available via FTP, over the internet at whitechapel.media.mit.edu/Vistex. Other studies should not be overlooked, as they may be of some help in ascertaining the utility of future investigations by providing us with some standard textures.

References

1. Haralick, R.M. and L.G. Shapiro, "Glossary of Computer Vision Terms," *Pattern Recognition*, **24**, 1, p 69-93, 1991.

2. Welsh, J.P., et al., *"SWOE JT&E Final Report,"* U.S. Army Cold Regions Research and Engineering Laboratory, Hanover, NH, 1995.

3. Bleiweiss, M.P., et al., *Appendix E of the SWOE JT&E Final Report - Analysis of SWOE JT&E Infrared Imagery and Environmental Data*, U.S. Army Cold Regions Research and Engineering Laboratory, Hanover, NH, 1995.

4. Fournier, A., D. Fussell, and L. Carpenter, "Computer Rendering of Stochastic Models," *Communication of the ACM*, **25**, 6, p 371-384, 1992.

5. Saupe, D., "Algorithms for Random Fractals," *The Science of Fractal Images*, eds. Pietgen, H.O. and D. Saupe, Springer-Verlag, Berlin/New York, 1988.

6. Hilgers, J.W., W.R. Reynolds, D.E. Strang, and J.R. McManamey, "Correlation Length Used as a Predictor of Fundamental Scale Lengths of Image Characterization", *Optical Engineering*, **35**, 3, p 786-793, 1996.

7. Austin, R.T., A.W. England, and G.H. Wakefield, "Special Problems in the Estimation of Power-Law Spectra as Applied to Topographical Modeling," *IEEE Transactions on Geoscience and Remote Sensing*, **32**, 4, p 928-939, 1994.

8. Fox, C.G., "Empirically Derived Relationships between Fractal Dimension and Power Law Form Frequency Spectra," *PAGEOPH*, **131**, p 211-239, 1989.

9. Stoksik, M.A., R.G. Lane, and D.T. Nguyen, "Practical Synthesis of Accurate Fractal Images," *Graphical Models and Image Processing*, **57**, 3, p 206-219, 1995.

10. Lohmann, G., "Analysis and Synthesis of Textures: A Co-Occurrence-Based Approach," *Comput. & Graphics*, **19**, 1, p 29-36, 1995.

11. Koenig, G.G., *Appendix G of the SWOE JT&E Final Report - Analysis of SWOE JT&E Infrared Imagery and Environmental Data*, U.S. Army Cold Regions Research and Engineering Laboratory, Hanover, NH, 1995.

12. Beale, R. and T. Jackson, *Neural Computing: An Introduction*, Adam Hilger, New York, 1991.

13. Sabol, B.M. and L.K. Balick, *"Generating Textures for Synthetic Thermal Scenes,"* 91-3, U.S. Army Cold Regions Research and Engineering Laboratory, Hanover, NH, 1991.

14. Press, W.H., S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in FORTRAN: the art of scientific computing, Second Edition,* Cambridge University Press, New York, 1992.

15. Schott, J.R, C.N. Salvaggio, S.D. Brown, R.A. Rose, "Incorporation of Texture in Multispecral Synthetic Image Generation Tools," Presented at the SPIE Aerosense, Targets and Backgrounds: Characterization and Representation, Col 2468, 23, Orlando, FL, 1995.

16. Choe, Y. and R.L. Kashyap, *Modeling, Estimation, and Pattern Analysis of Random Texture on 3-D Surfaces*, TR-EE-91-4, School of Electrical Engineering, Purdue University, West Lafayette, IN, 1991.

Acronyms and Abbreviations

AR	autoregressive
GLCM	grey-level co-occurrence matrix
IR	infrared
IRMA	infrared modeling and analysis
SAR	synthetic aperture radar
SWOE JT & E	Smart Weapons Operability Environment Joint Test and Evaluation
WES	Waterways Experiment Station

Appendix A

Modified Mid-Point Displacement Algorithm

The modified mid-point displacement algorithm is a method of generating a two dimensional fractional Brownian "motion". This method is described in Stoksik et al. (1995). Much of the notation and wording of the following description comes from this paper.

A two dimensional fractional Brownian motion (fBm) is a random process with an average spectral power density given by

$$S(f_1, f_2) = \frac{\sigma^2}{\left(\sqrt{f_1^2 + f_2^2}\right)^{\beta}} \qquad \text{(A-1)}$$

where σ^2 is a constant. fBm is a zero-mean nonstationary Gaussian random process $B_H(\bar{t})$ indexed by the parameter $H, 0 < H < 1$. The symbol $\bar{t}$ represents the vector pair (t_1, t_2).

The exponent β, H, and the fractal dimension δ_f are related by the equations:

$$\beta = 2H + 2 \qquad \text{(A-2)}$$

$$\delta_f = 3 - H \qquad \text{(A-3)}$$

The statistical behavior of fBm can also be described by its covariance expression,

$$E\{B_H(\bar{t})B_H(\bar{s})\} = \sigma^2 \left(|\bar{t}|^{2H} + |\bar{s}|^{2H} - |\bar{t} - \bar{s}|^{2H}\right) \qquad \text{(A-4)}$$

where, for instance,

$\bar{t} = (t_1, t_2)$
$|\bar{t}| = $ length of $\bar{t}$

and $\bar{s}$, $|\bar{s}|$, and $|\bar{t} - \bar{s}|$ are defined similarly.

$B_H(\bar{t})$ is a nonstationary process with stationary increments. From Equation (A-4) we can derive the following equation asserting the finiteness of the structure function $D(\bar{p})$:

$$D(\bar{p}) = \mathrm{E}\left\{\left[B_H(\bar{t} + \bar{p}) - B_H(\bar{t})\right]^2\right\} = 2\sigma^2 |\bar{p}|^{2H} \qquad\qquad \text{(A-5)}$$

for finite p. In this expression it is evident that the middle expression involving the expectation operator E is independent of t. The purpose of the modified random displacement algorithm is to generate a sampled grid of points where the statistical interrelationship of the points is given by Equation (A-5), the structure function condition. Assuming a sampled set of points are available which satisfy Equation (A-5), the algorithm generates new points between the existing points. These new points are generated so as to ensure that they have the correct statistical relationship with their nearest neighbors.

The originally proposed random midpoint displacement algorithm (RMDA) is only accurate for the case of simple Brownian motion ($H = 0.5$). For other values of H there are correlations between the existing random variables that must be considered when calculating the displacements for the interpolated samples. In order to generate accurate fBm with the correct structure function, the algorithm must be modified to account for the correlations between points within the fBm.

The modified mid-point displacement algorithm begins by finding the corner values of a two dimensional array and then stage by stage finds the values at other points in the manner illustrated by Figures A-1 and A-2. A broad view of the algorithm is gained by examining the organization of the computer code *accfrac.c* (Appendix I). The computer code shows that each stage of the algorithm determines (1) some interior points, (2) the edge points, (3) some more interior points, and then (4) interior points which are at positions symmetric to the diagonal relative to those determined in (3).

The modified algorithm starts with the calculation of the four corner samples denoted by A, B, G, and D in Figure A-3. If these four corner variates satisfy Equation (A-5) we have the following six equations corresponding to Equation (A-5) applied to the six possible combinations:

63

$$E\{(A-B)^2\} = 2\sigma^2 d^{2H} \qquad\qquad\qquad (A\text{-}6)$$

$$E\{(A-G)^2\} = 2\sigma^2 d^{2H} \qquad\qquad\qquad (A\text{-}7)$$

$$E\{(B-D)^2\} = 2\sigma^2 d^{2H} \qquad\qquad\qquad (A\text{-}8)$$

$$E\{(G-D)^2\} = 2\sigma^2 d^{2H} \qquad\qquad\qquad (A\text{-}9)$$

$$E\{(A-D)^2\} = 2\sigma^2 (d\sqrt{2})^{2H} \qquad\qquad (A\text{-}10)$$

$$E\{(B-G)^2\} = 2\sigma^2 (d\sqrt{2})^{2H} \qquad\qquad (A\text{-}11)$$

The last two equations reflect the fact that the distance between the corner sample locations is $d\sqrt{2}$.

We assume that the values A, B, G, and D are derived from the following combination of the auxiliary variates, S, T, U, V, W, and X:

$$A = S + W/2 \qquad\qquad\qquad (A\text{-}12)$$
$$B = T + X/2 \qquad\qquad\qquad (A\text{-}13)$$
$$G = U - X/2 \qquad\qquad\qquad (A\text{-}14)$$
$$D = V - W/2 \qquad\qquad\qquad (A\text{-}15)$$

$$
\begin{array}{ccc}
A & 4 & B \\
2 & 1 & 3 \\
G & 5 & D
\end{array}
\left\{
\begin{array}{l}
\text{interior: } 1 \\
\text{edge: } 2,3,4,5 \\
\text{interior: none} \\
\text{interior: none}
\end{array}
\right.
$$

Figure A-1. This figure illustrates the values found in stage 1. Only the center point labeled 1 and the edge points 2 through 5 are determined in stage 1. No interior points are determined in parts (3) and (4) of this stage.

```
A   12   4   16   B     ⎧ interior:  6,7,8,9
10   6   20   8   11    ⎪
 2  18   1   19   3     ⎨ edge:  10-17
14   7   21   9   15    ⎪ interior:  18,19
 G  13   5   17   D     ⎩ interior:  20,21
```

Figure A-2. This figure shows the result of stage 2. In part (1) of stage 2 the points 6,7,8, and 9 are found in that order. In (2), the edges 10 through 17. In part (3) the points 18 and 19 are found. Finally points 20 and 21 are determined.

```
A•    e•   B•
      •  m•    •
G•       •  D•
```
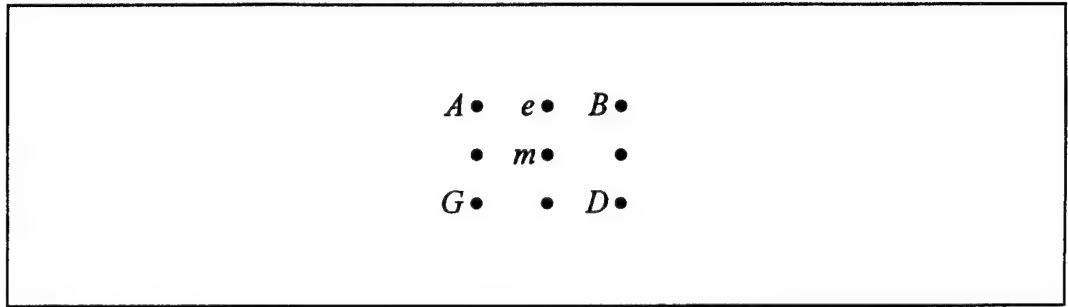
Figure A-3. Graphical aid to assist in defining the detailed steps of the Modified Mid-point displacement Algorithm.

Here S, T, U, and V are zero mean Gaussian variates with the same variance σ_c^2 yet to be determined. Both W and X are also zero mean Gaussian variates with the different variance σ_d^2 also yet to be determined. The variates W and X force correlations between the corner pairs (A,D) and (B,G).

Equations (A-6) through (A-11) force conditions on the two variances of the six auxiliary variables which can be solved to yield the following expressions for the values of σ_c^2 and σ_d^2 :

$$\sigma_c^2 = \sigma^2 \left(2 - 2^H\right) d^{2H} \qquad (A-16)$$

$$\sigma_d^2 = 4\sigma^2 \left(2^H - 1\right) d^{2H} \qquad (A-17)$$

The modified mid-point displacement algorithm uses two recursive steps. One is used to find the value of an interior center point m as in Figure A-3. The other is used to find the value of an edge point also labeled e in Figure A-3. The labels found in Figure A-3 will be used in the discussions that follow which explain the respective recursive steps.

In Figure A-3, the value m at the center of a square with corner values A, B, G, and D is determined by the following formula:

$$m = \frac{(A+B+G+D)}{4} + \varepsilon \qquad \text{(A-18)}$$

where ε is an independent Gaussian variate with zero mean and variance $\text{Var}(\varepsilon)$. The value of $\text{Var}(\varepsilon)$ is obtained as follows. From Equations (A-12) through (A-18), we can determine

$$E\{(A-m)^2\} = \sigma^2\left(2^{H-2} + \frac{1}{2}\right)d^{2H} + \text{Var}(\varepsilon) \; ; \qquad \text{(A-19)}$$

whereas that predicted from Equation (A-5), the structure function condition, is

$$E\{(A-m)^2\} = 2\sigma^2\left(\frac{d}{\sqrt{2}}\right)^{2H} . \qquad \text{(A-20)}$$

The variance of the displacement, $\text{Var}(\varepsilon)$, is, therefore,

$$\text{Var}(\varepsilon) = \sigma^2\left(2^{1-H} - 2^{H-2} - \frac{1}{2}\right)d^{2H} . \qquad \text{(A-21)}$$

This value for $\text{Var}(\varepsilon)$ will also force Equation (A-5) to hold for the other three pairs (B,m), (G,m), and (D,m). By comparison, the unmodified RMDA takes no account of the correlations of the corner samples and would use the variance $\sigma^2(2^{-H})d^{2H}$.

Using the notation of Figure A-3, the value e at an edge is found by:

$$e = \frac{(A+B)}{2} + \eta \ , \qquad\qquad\qquad (A\text{-}22)$$

where η is an independent Gaussian variate with zero mean and variance

$$\mathrm{Var}(\eta) = \sigma^2 \left(2^{1-2H} - \frac{1}{2} \right) d^{2H} \ . \qquad\qquad (A\text{-}23)$$

This value for $\mathrm{Var}(\eta)$ is found through evaluation of $E\{(A\text{-}e)^2\}$ using the definition of e and the structure function condition. By way of contrast, the RMDA would assign the following value for the variance of η:

$$\mathrm{Var}(\eta) = \sigma^2 \left(\frac{d}{2} \right)^{2H} = \sigma^2 \left(2^{-2H} \right) d^{2H} \ . \qquad (A\text{-}24)$$

In Stoksik, et al (1995), it is shown that this method yields images whose structure functions are a good approximation to the ideal structure functions. The structure functions were calculated by radial averaging of the two-dimensional image (surface) and then plotting the resultant one-dimensional curve on a log-log plot. It is not perfect because each interpolated point is calculated from only the four adjacent points surrounding it and consequently the structure function between an interpolated point and a more distant point is not necessarily exact. Figure A-4 demonstrates the results of using this algorithm for the values of $H = 0.1$ and 0.9, respectively. Figure A-5 shows similar results for the original mid-point displacement algorithm as discussed in the main body of this report.
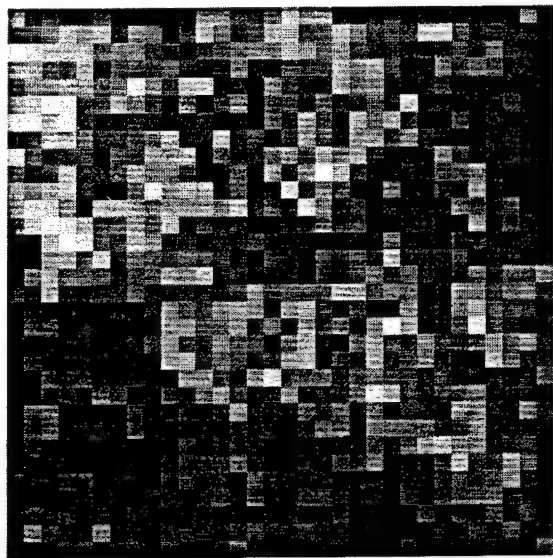
H = 0.1   H = 0.9

Figure A-4. Texture maps from the modified mid-point displacement algorithm for the parameters H=0.1 and =0.9.
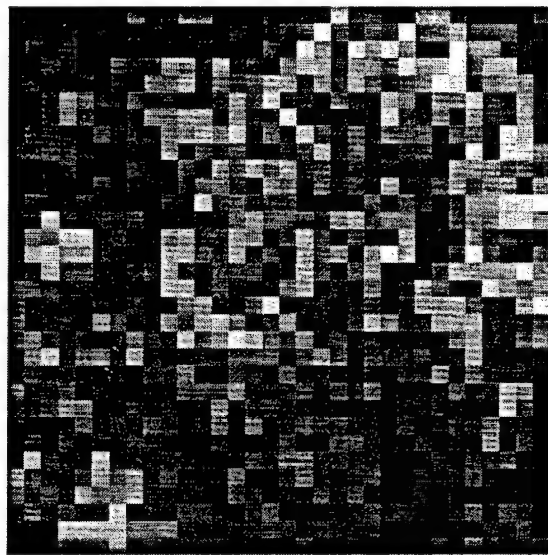
H = 0.1
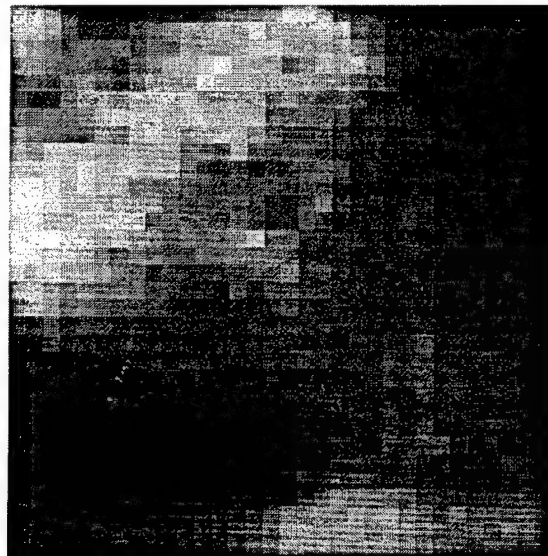
H = 0.9

Figure A-5. Texture maps from the original mid-point displacement algorithm for the parameters H=0.1 and =0.9.
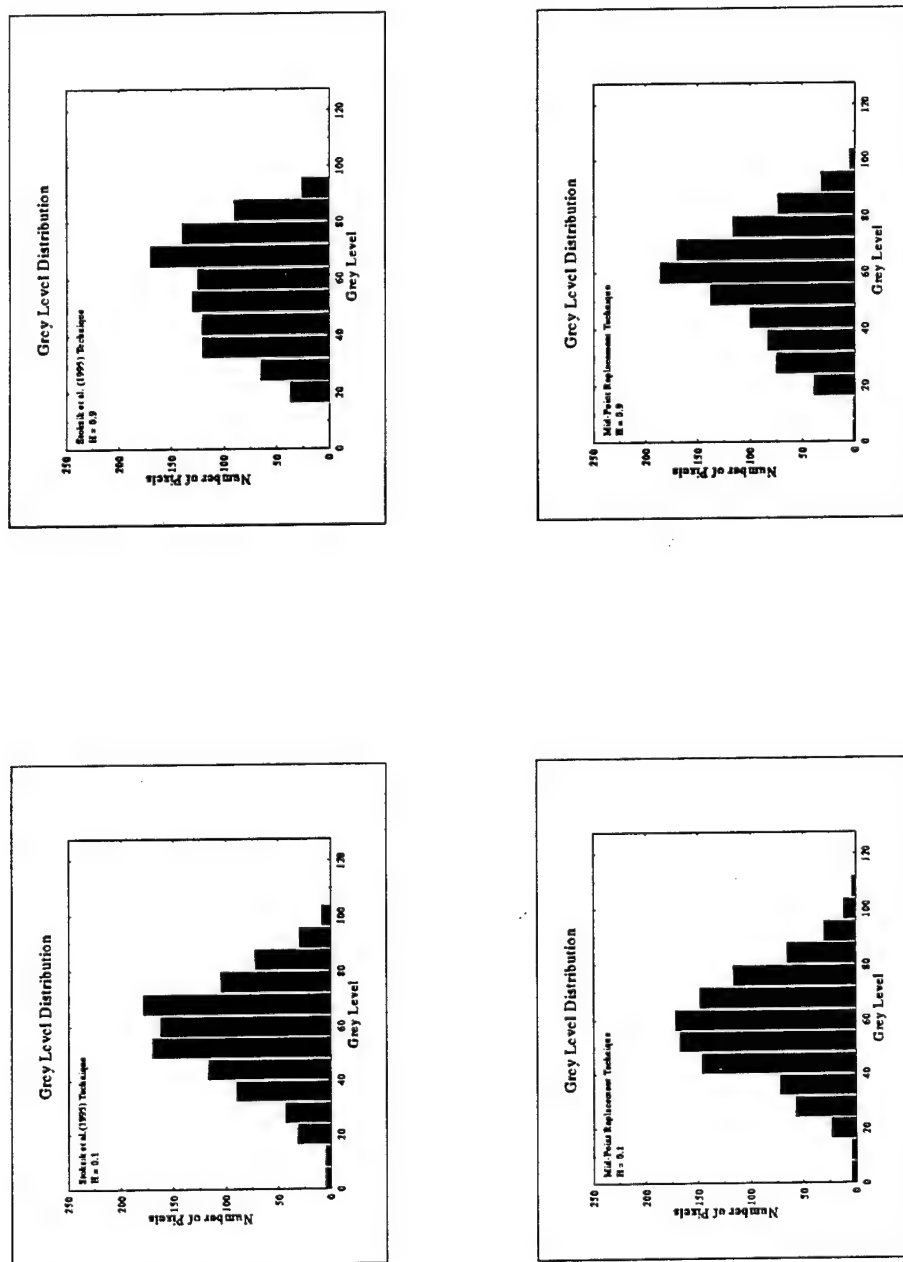
Figure A-6. Histograms of grey levels for the images of Figures A-4 and A-5, respectively. The upper plots are for the modified mid-point displacement method and the lower panels are for the original mid-point displacement.

Reference: Appendix A

Stoksik, M.A., R.G. Lane, and D.T. Nguyen, "Practical Synthesis of Accurate Fractal Image," *Graphical Models and Image Processing*, **57** , 3, p 206-219, 1995.

Appendix B

Simulated Annealing: A Brief Discussion

The primary references used in the following brief discussion are Kirkpatrick, et al. (1983), Kirkpatrick (1984), and Metropolis (1953). The McGraw-Hill Dictionary of Scientific and Technical Terms (Parker 1989) defines *annealing* in the following way:

"To treat a metal, alloy, or glass with heat and then cool to remove internal stresses and to make the material less brittle."

Simulated annealing is a process whereby a "system" is brought to a "reduced energy state", or "temperature" through emulation of a naturally occurring process -- this is accomplished by a variety of algorithms. The goal, in our situation, is to find an absolute minimum through a process that does not get led astray by local minima nor is caught for good in a local minimum -- we want to find a rearrangement of pixels such that the resulting image has the same, or nearly the same, GLCM as the target arrangement of pixels. To accomplish this goal, simulated annealing is the process of choice.

In metallurgy, a system -- a metal -- is heated to the fluid state, and the temperature at which that occurs is noted. The temperature is then reduced, according to some schedule (the "annealing schedule") until the metal has cooled to some predetermined state -- usually such that the end state is of uniform condition; i.e., no cracks, etc. For the statistical mechanics analogue, a system of particles is studied at "low temperature" which state has been achieved by some annealing process. What happens in this case, is that the system is at some initial configuration, $\{x_i\}$, with some potential energy E which is determined by the Boltzmann probability factor:

$$\exp\left[-E(\{x_i\}) / kT\right] .$$

For our purposes, instead of the energy state, we use a "cost factor" defined to be Δ and, instead of just decreasing T to arrive at an end state, we use what is known as the "Metropolis Algorithm" to keep the iterative process moving. This algorithm simulates the behavior of a many-body system at some temperature. Use of this algorithm then "provides a natural tool for bringing the techniques of statistical mechanics to bear on optimization." Still quoting from Kirkpatrick, et al. 1984:

Iterative improvement, commonly applied to such problems, is much like the microscopic rearrangement processes modeled by statistical mechanics, with the cost function playing the role of energy. However, accepting only rearrangements that lower the cost function of the system is like extremely rapid quenching from high temperatures to $T = 0$, so it should not be surprising that resulting solutions are usually metastable. The Metropolis procedure from statistical mechanics provides a generalization of iterative improvement in which controlled uphill steps can also be incorporated in the search for a better solution.

The simulated annealing process consists of first 'melting' the system being optimized at a high effective temperature, then lowering the temperature by slow stages until the system 'freezes' and no further changes occur. At each temperature , the simulation must proceed long enough for the system to reach a steady state. The sequence of temperatures and the number of rearrangements of the $\{x_i\}$ attempted to reach equilibrium at each temperature can be considered an annealing schedule.

Annealing, as implemented by the Metropolis procedure, differs from iterative improvement in that the procedure need not get stuck since transitions out of a local optimum are always possible at nonzero temperature.

The monograph by Beale and Jackson (1991) discusses this process as applied to what are known as "Hopfield Networks" -- a neural network which "...consists of a number of nodes, each connected to every other node: it is a fully-connected network...". The Hopfield net uses simulated annealing to allow exit from local minima -- large changes are implemented at "high" temperatures in search of a path to a global minimum. The temperature is reduced as the process proceeds which allows less drastic changes to be used with the more probable result that the process ends up in a lower energy state. The probability distribution which controls this process is the Boltzmann distribution.

References: Appendix B

Beale, R. , and T. Jackson, *Neural Computing: An Introduction*, Adam Hilger, New York, 1991.

Kirkpatrick, S., C.D. Gelatt, Jr., M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, **220**, 4598, p 671-680, 1983.

Kirkpatrick, S., "Optimization by Simulated Annealing: Quantitative Studies," *Journal of Statistical Physics*, **34**, 5 and 6, p 975-986, 1984

Metropolis, N., A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller, "Equation or State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, **21** 6, p. 1087-1092, 1953.

Parker, S.P., *McGraw-Hill Dictionary of Scientific and Technical Terms, 4th ed.,* McGraw-Hill, New York, 1989.

Appendix C

Grey Level Co-Occurrence Matrix: A Brief Discussion

Grey-level co-occurrence analysis "... characterizes the micro-texture of an image region by measuring the dependence between pairs of grey levels (our grey levels are in units of apparent temperature) arising from pixels in a specified spatial relation" [Haralick and Shapiro, 1991]. This technique may be applied to either the whole image (global feature definition) or to a ROI (local feature definition). Although there are an extremely large number of parameters to be derived from this analysis, it can be limited to a few that have been shown to provide good discrimination between various textures [Wahl, 1987; Ballard and Brown, 1982; Marceau et al.,1990]:

1) contrast -- this parameter is greatest when adjacent pixels are very different in grey level

2) homogeneity -- this is greatest when most of the co-occurrences are for the same two grey levels, it is a measure of how consistently a pattern is repeated in an image

3) entropy -- this measures the "information content of the image; it is greatest when all co-occurrence possibilities occur in equal proportion

4) correlation -- correlation is greatest when the pixels in the (i,j) pair are similar in grey level value and both values at the same time are either above or below the average values for their respective positions in the pair (μ_x and μ_y)

Other "texture" analysis techniques are also available [He and Wang, 1990; Therrien et al., 1986].

The grey-level co-occurrence matrix (GLCM) is a square matrix of width m based on the number of grey levels in the image under analysis. For example, an image quantized at 256 levels causes a GLCM of 256 x 256 to be formed. Because this creates an extremely large matrix, the number of levels is generally reduced to, generally, 8 or 16 [Wahl, 1987]. In addition, as will be seen shortly in the definition of how the GLCM is formed, it is possible to create a matrix for a large number of angles or orientations as well; this too is usually reduced to a small number: 0°, 45°, 90°, and 135°.

The matrix is then a "distribution" of the number of times that certain configurations of brightness levels occur. This can best be explained by example [Haralick, 1974]:

digital image (l columns and k rows):

$$l = 1 \ 2 \ 3 \ 4$$

$$
\begin{array}{c c c c c}
k = 1 & 0 & 0 & 1 & 1 \\
2 & 0 & 0 & 1 & 1 \\
3 & 0 & 2 & 2 & 2 \\
4 & 2 & 2 & 3 & 3 \\
\end{array}
$$

generalized GLCM:

	grey level			
	0	1	2	3
0	#(0,0)	#(0,1)	#(0,2)	#(0,3)
1	#(1,0)	#(1,1)	#(1,2)	#(1,3)
grey level				
2	#(2,0)	#(2,1)	#(2,2)	#(2,3)
3	#(3,0)	#(3,1)	#(3,2)	#(3,3)

where #(i,j) is the number of times that the i,j pair occur in the digital image in a particular orientation. Specifically, for $\phi=0°$ (horizontal orientation), there are four times that a zero grey level value occurs left/right, right/left in the digital image (in some instances, only the left/right or only the right/left; i.e., uni-directional, direction is chosen which would yield a value of 2 for this case):

k,l = 1,1 is next to 1,2
 = 1,2 is next to 1,1
 = 2,1 is next to 2,2
 = 2,2 is next to 2,1.

There are no more k,l positions where a zero grey level occurs and right next to it, another zero occurs. This process continues in this manner to yield the 0° GLCM :

And, we can continue with the example so that $\phi = 45°$, 90°, and 135° are processed:

Similar matrices can be obtained for distance measures greater than the value of 1 which we have used -- we could have, for example, determined the number of times that a zero is 2 spaces from another zero, etc.

The "textural features" that can be determined from these GLCM are at least 28 in number; however, there are only 4-6 that have been found to be most useful (as stated above). The mathematical formulations and the labeling used in the table headings found in the main body of the report, for some of these, are given below (these are the ones which we have found to be most useful):

homogeneity or energy or angular second moment:

$$f_1 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} \left(\frac{P(i,j)}{R} \right)^2$$

contrast or inertia:

$$f_2 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} (i-j)^2 \frac{P(i,j)}{R}$$

80

correlation:

$$f_3 = \frac{\displaystyle\sum_{i=1}^{N_g}\sum_{j=1}^{N_g}(i - \mu_x)(j - \mu_y)\frac{P(i,j)}{R}}{\sigma_x \sigma_y}$$

entropy:

$$f_9 = \sum_{i-1}^{N_g}\sum_{j=1}^{N_g}\frac{P(i,j)}{R}\log\left[\frac{P(i,j)}{R}\right]$$

and the definitions for the variables in the above equations are:

$P(i,j)$ = the value of the i, j element of the GLCM,

$R \equiv$ normalization constant (equal to the number of neighboring resolution cell pairs used in computing the matrix),

$N_g \equiv$ number of gray levels,

$$\mu_x = \sum_{i=1}^{N_g} i \sum_{j=1}^{N_g} P(i,j),$$

$$\mu_y = \sum_{i=1}^{N_g} j \sum_{i=1}^{N_g} P(i,j),$$

$$\sigma_x^2 = \sum_{i=1}^{N_g}(i - \mu_x)^2 \sum_{j=1}^{N_g} P(i,j), \text{ and}$$

$$\sigma_y^2 = \sum_{j=1}^{N_g}(j - \mu_y)^2 \sum_{i-1}^{N_g} P(i,j).$$

References: Appendix C

Ballard, D.H. and C.M. Brown, *Computer Vision*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

Haralick, R.M. And L.G. Shapiro, "Glossary of Computer Vision Terms," *Pattern Recognition,* **24**, 1, p 69-93, 1991.

Wahl, F.M., *Digital Image Signal Processing*, Artech House, Boston, 1987.

Marceau, D.J., P.J. Howarth, J., M.M. Dubois, and D.J. Gratton, "Evaluation of the Grey-Level Co-Occurrence Matrix Method for Land-Cover Classification Using SPOT Imagery, *IEEE Transactions on Geoscience and Remote Sensing,* **28,** 4, p 513-519.

Appendix D

Histograms of Texture Metrics

Figure D-1. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "correlation length" as applied to the texture map generated by the Stoksik (1995) technique.

Figure D-2. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "correlation length" as applied to the texture map generated by the Lohmann (1995) technique.

Figure D-3. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "correlation length" as applied to the texture map generated by the Midpoint Replacement technique.

Figure D-4. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "correlation length" as applied to the texture map generated by the IRMA (smoothed output) technique.

85

Figure D-5. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "correlation length" as applied to the texture map generated by the IRMA (unsmoothed output) technique.

Figure D-6. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Homogeneity" as applied to the texture map generated by the Stoksik (1995) technique.

86

Figure D-7. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Homogeneity" as applied to the texture map generated by the Lohmann (1995) technique.

Figure D-8. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Homogeneity" as applied to the texture map generated by the Midpoint Replacement technique.
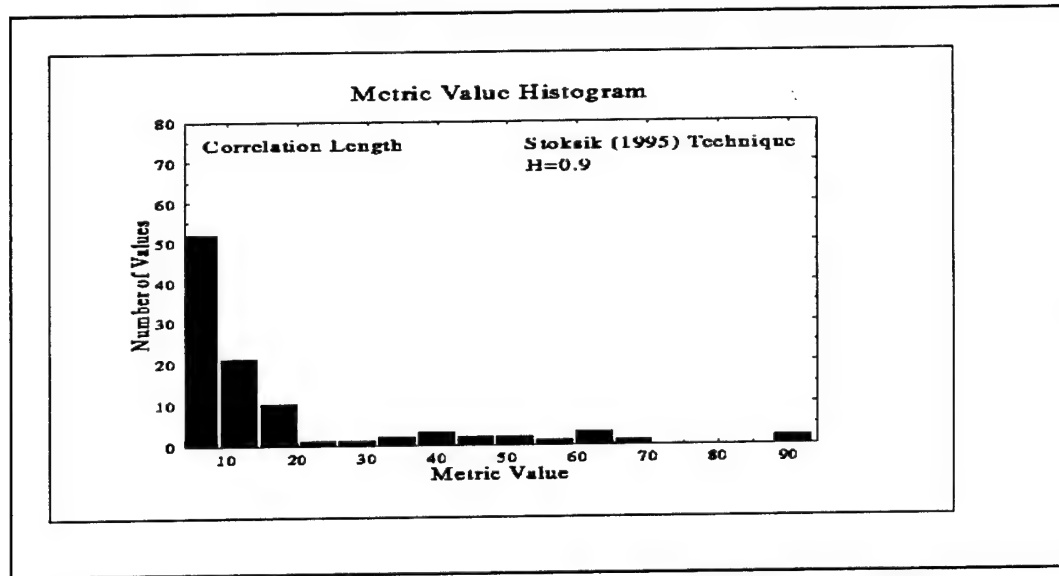
Figure D-9. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Homogeneity" as applied to the texture map generated by the IRMA (Smoothed Output) technique.
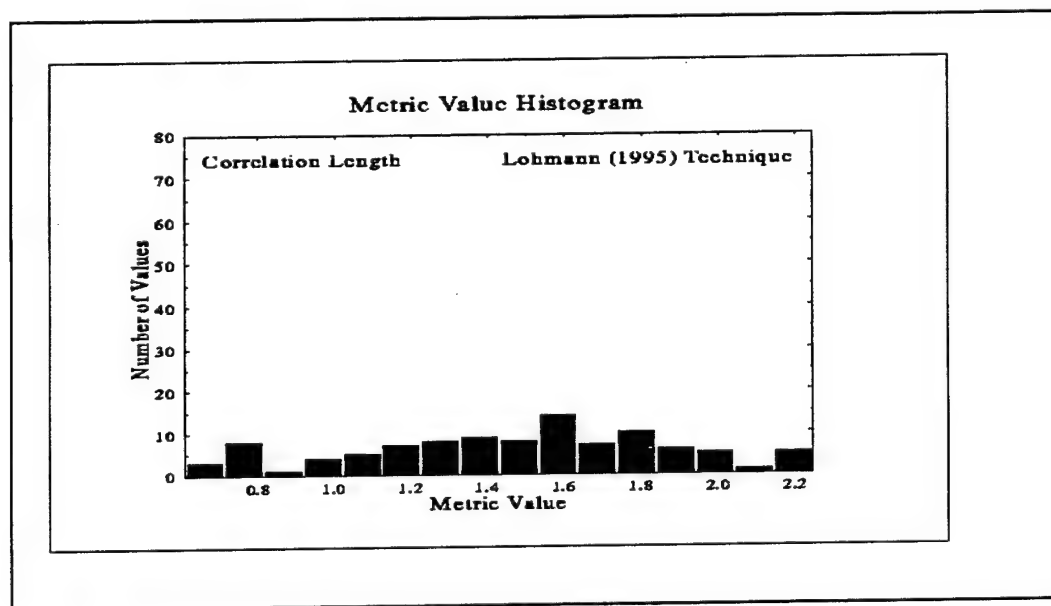

Figure D-10. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Homogeneity" as applied to the texture map generated by the IRMA (Unsmoothed Output) technique.
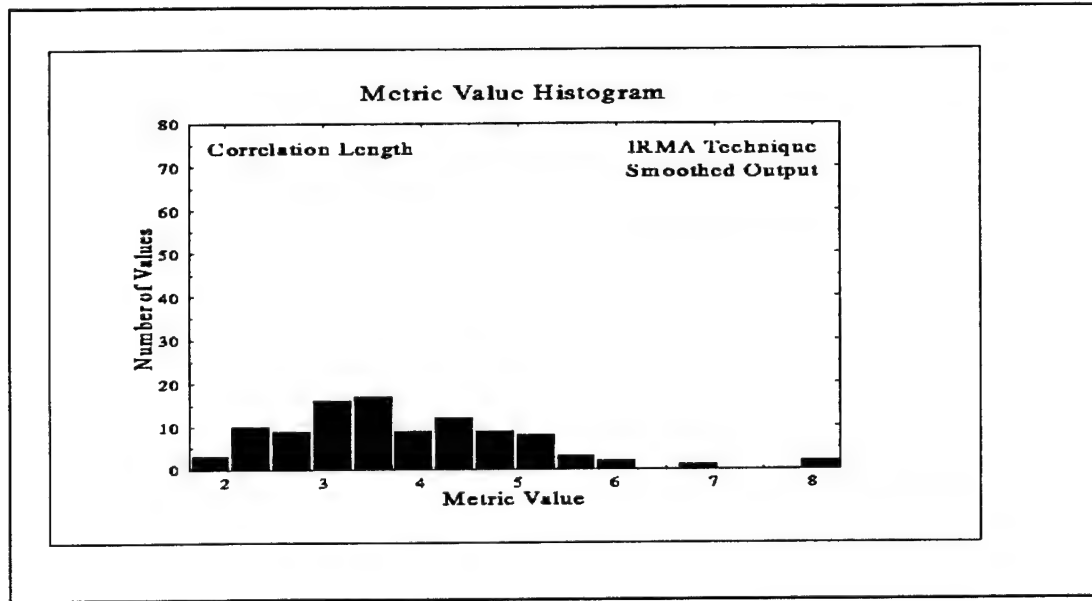
88

Figure D-11. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Correlation" as applied to the texture map generated by the Stoksik (1995) technique.

Figure D-12. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Correlation" as applied to the texture map generated by the Lohmann (1995) technique.
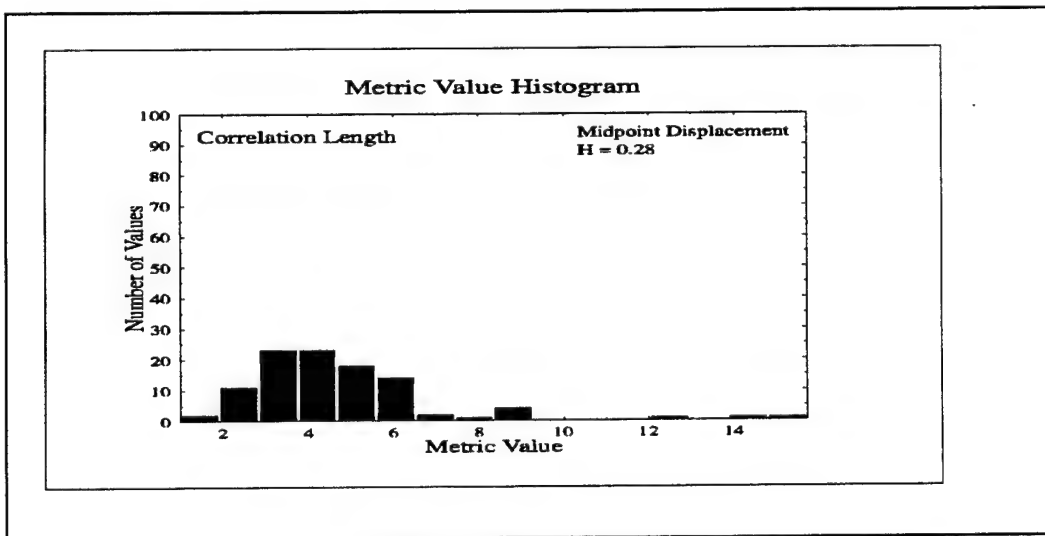
89

Figure D-13. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Correlation" as applied to the texture map generated by the Midpoint Replacement technique.
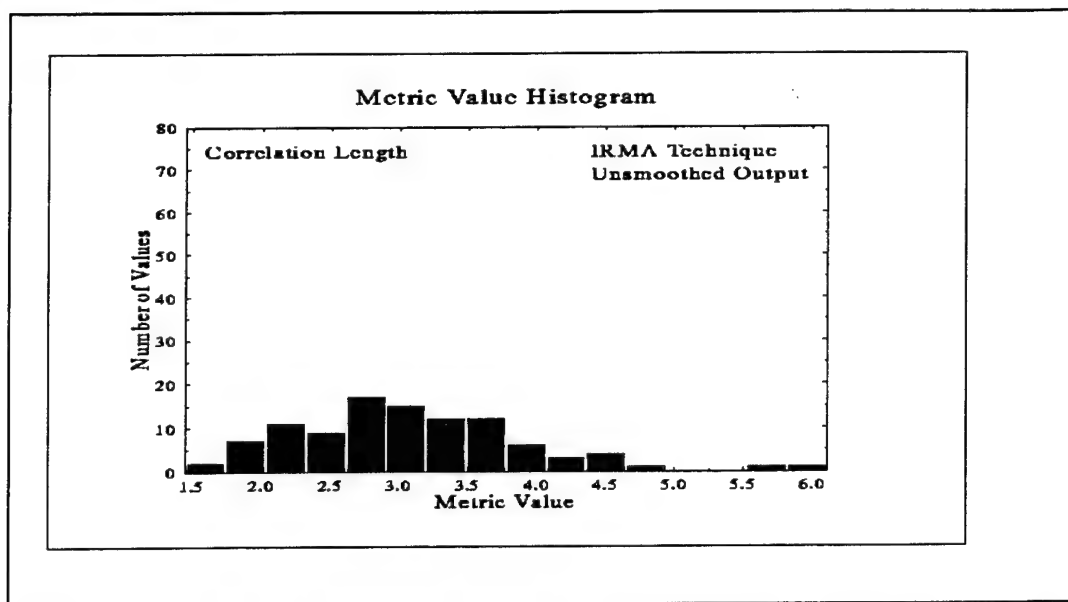

Figure D-14. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Correlation" as applied to the texture map generated by the IRMA (Smoothed Output) technique.

Figure D-15. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Correlation" as applied to the texture map generated by the IRMA (Unsmoothed Output) technique.
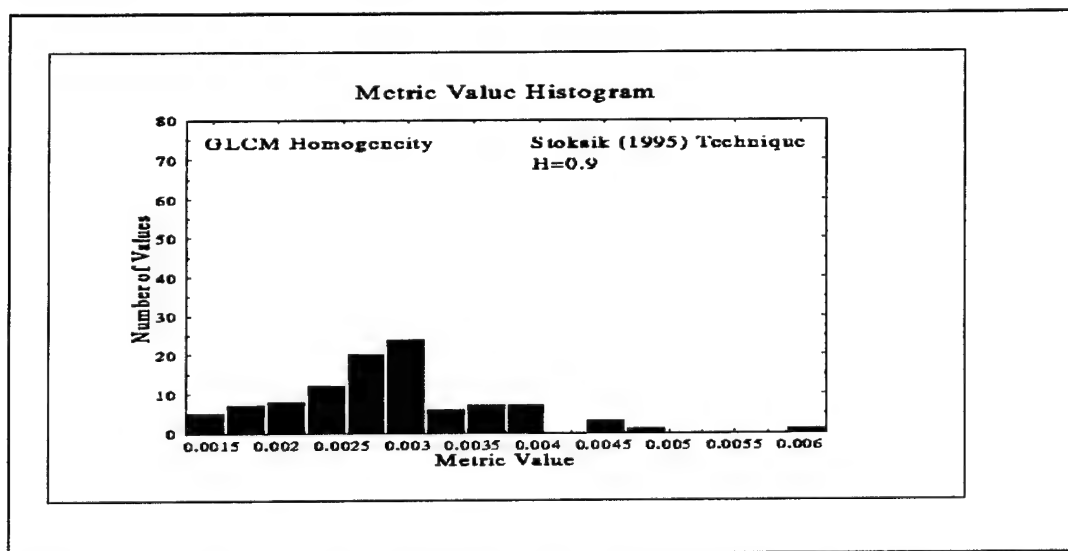

Figure D-16. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Entropy" as applied to the texture map generated by the Stoksik (1995) technique.
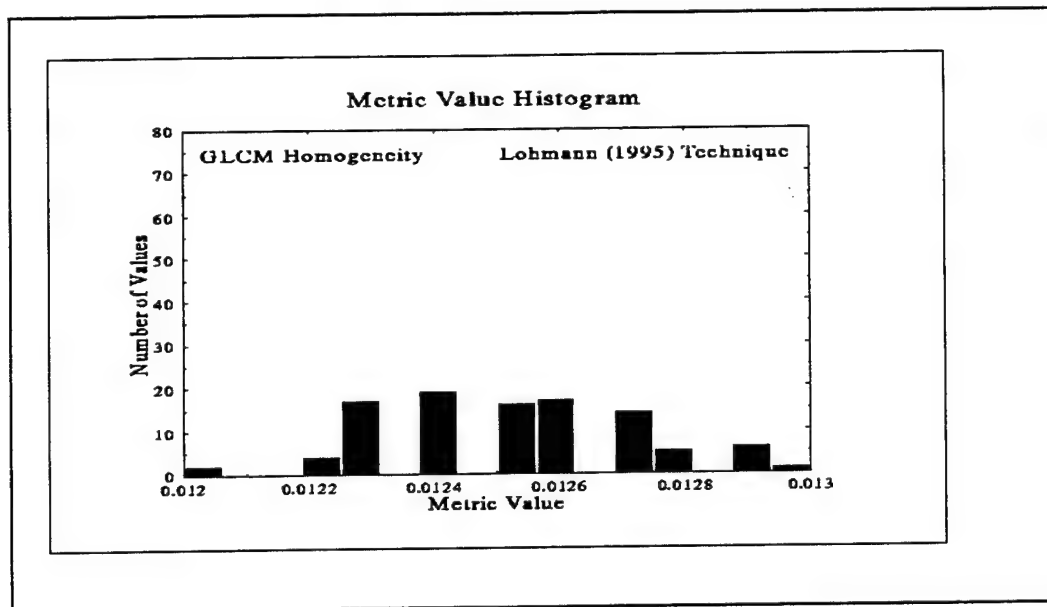
Figure D-17. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Entropy" as applied to the texture map generated by the Lohmann (1995) technique.
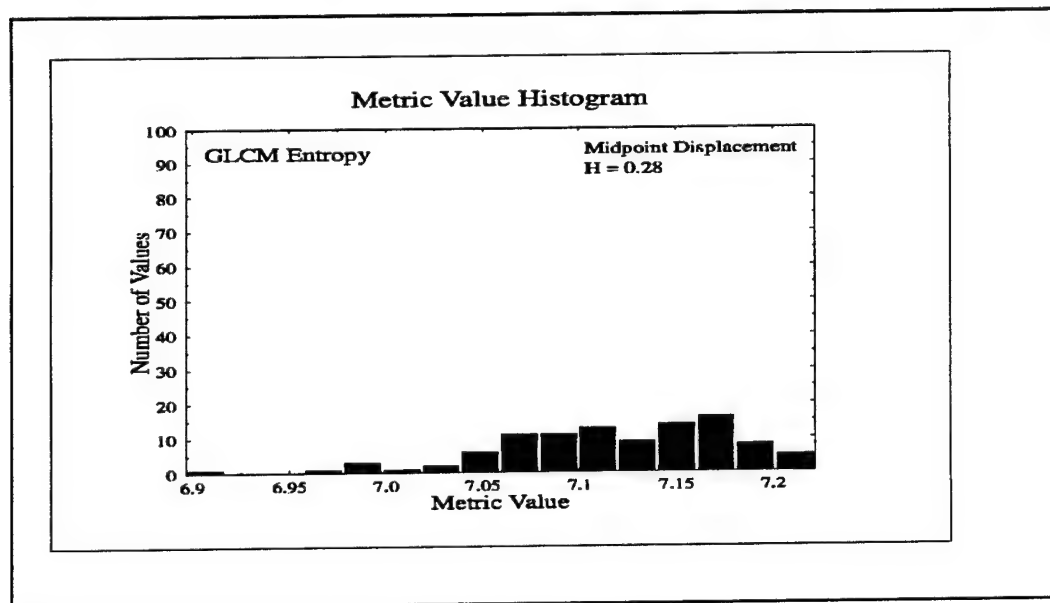
Figure D-18. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Entropy" as applied to the texture map generated by the Midpoint Replacement technique.
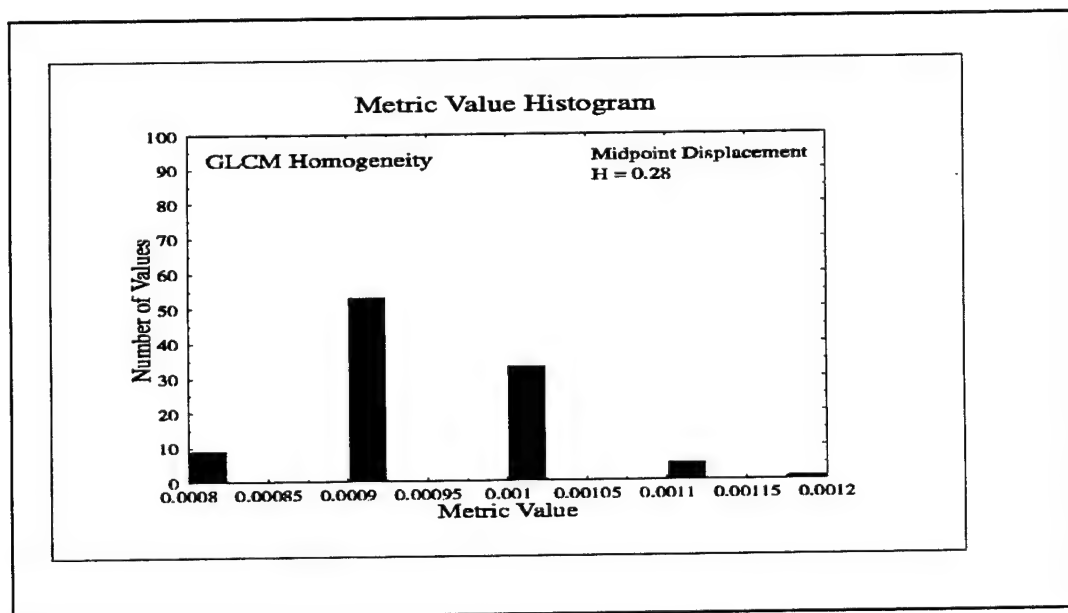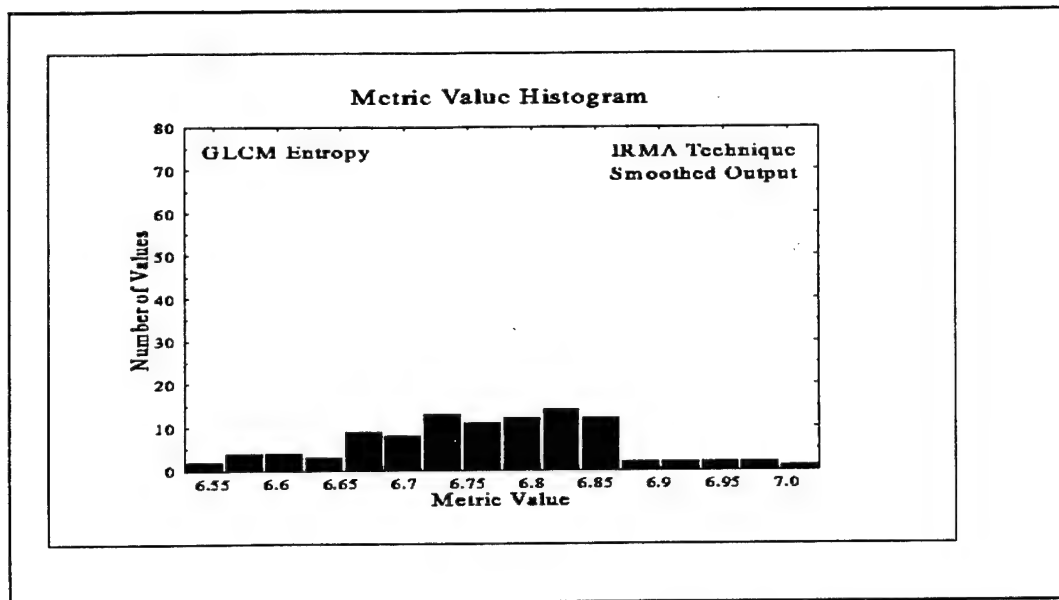
Figure D-19. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Entropy" as applied to the texture map generated by the IRMA (Smoothed Output) technique.
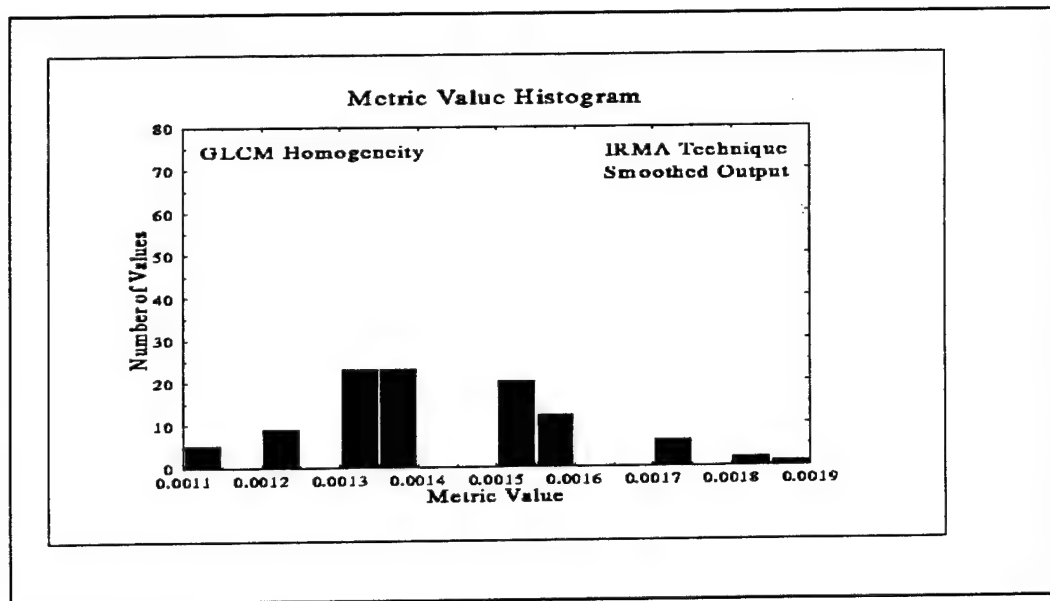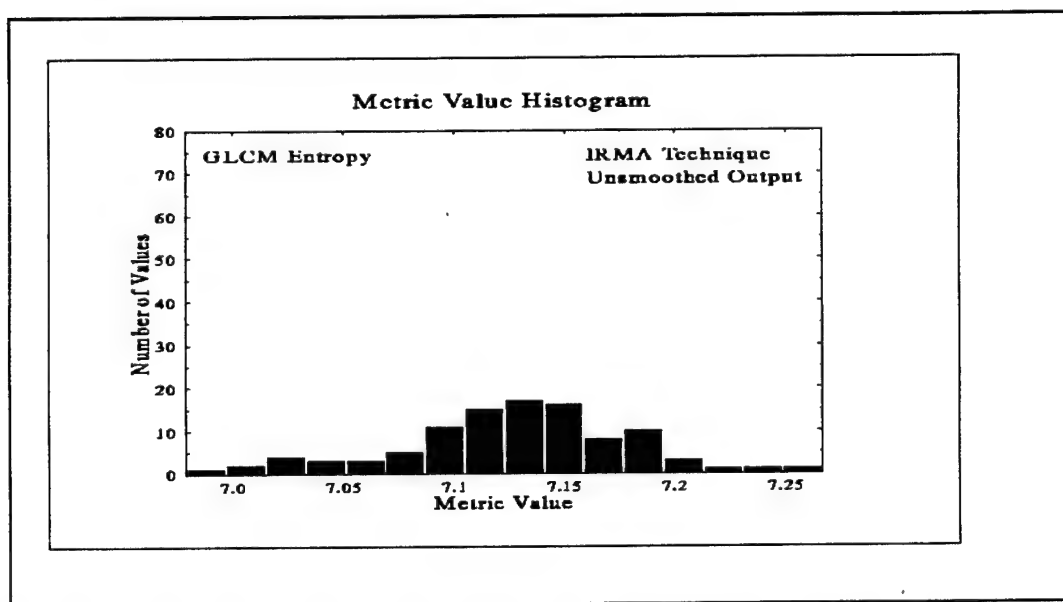

Figure D-20. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Entropy" as applied to the texture map generated by the IRMA (Unsmoothed Output) technique.
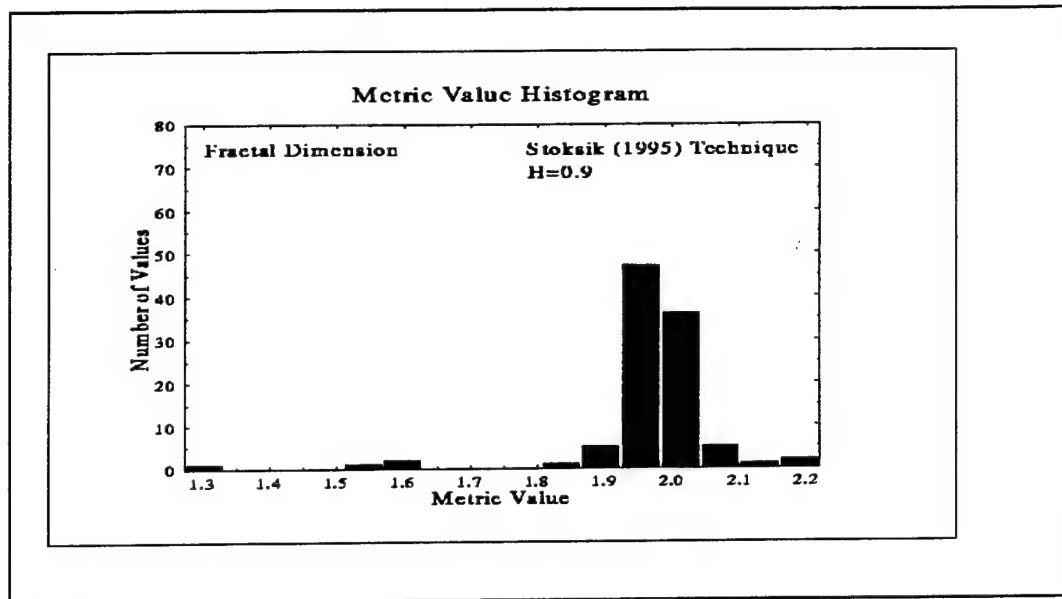
Figure D-21. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "Fractal Dimension" as applied to the texture map generated by the Stoksik (1995) technique.
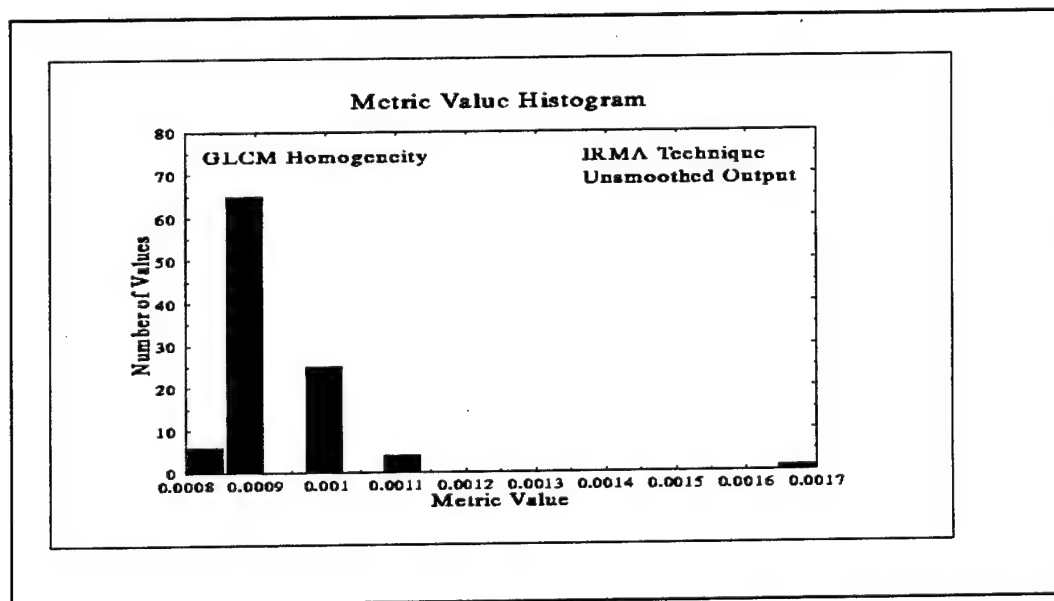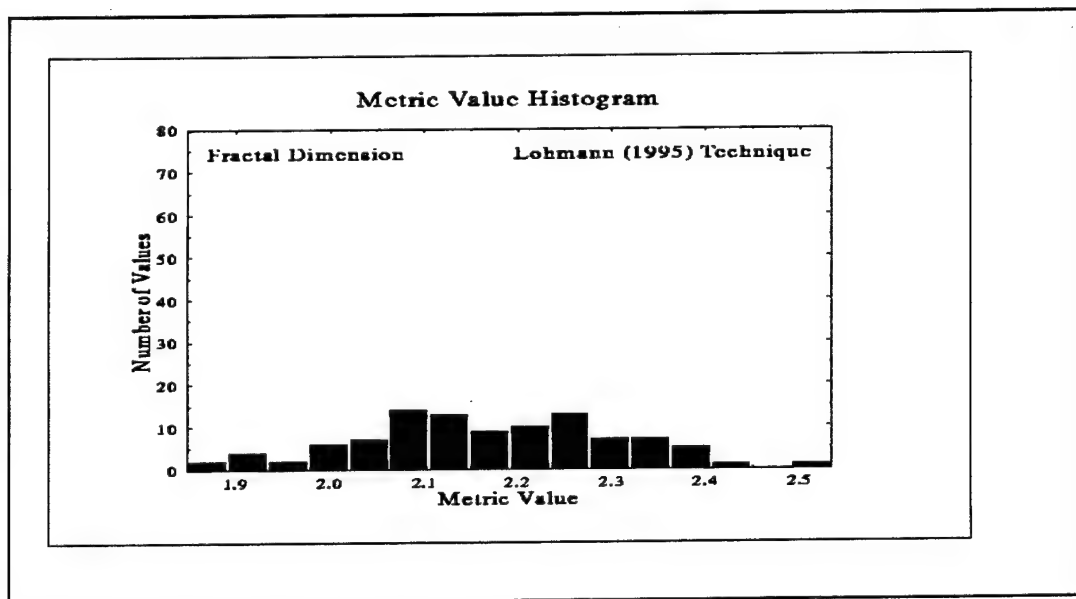

Figure D-22. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "Fractal Dimension" as applied to the texture map generated by the Lohmann (1995) technique.
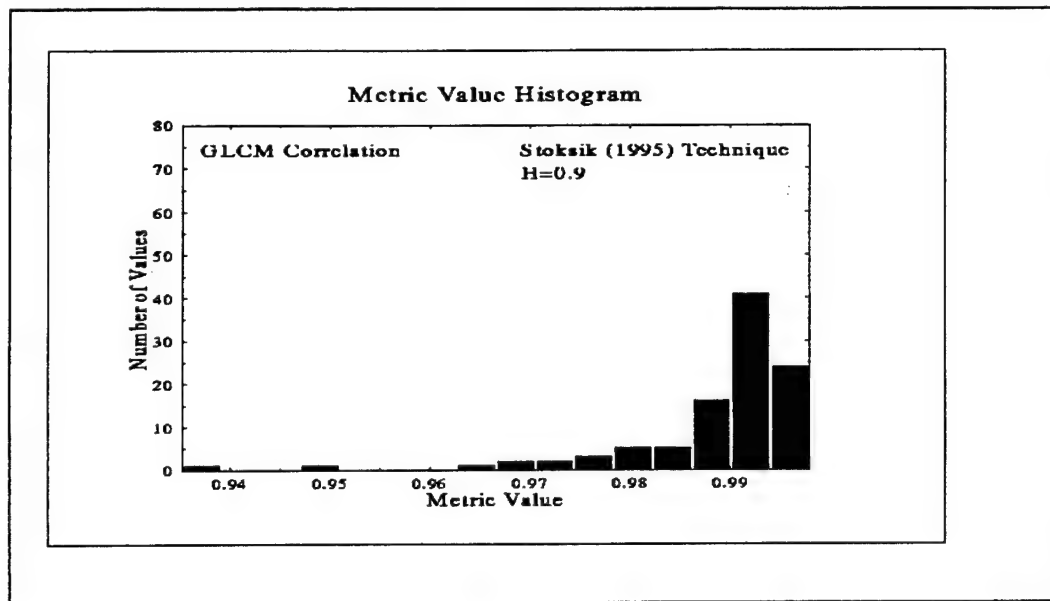
94
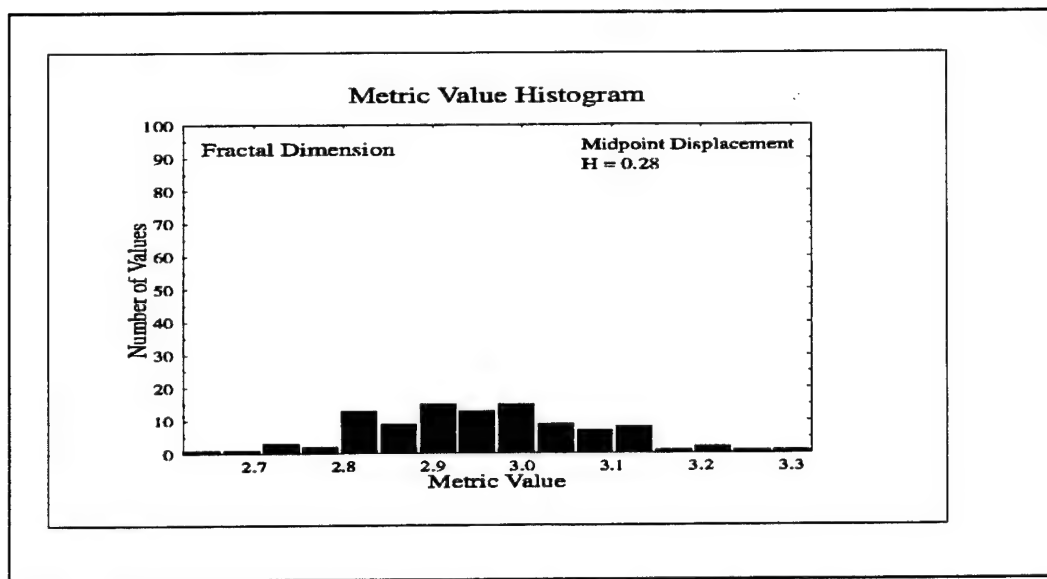
Figure D-23. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "Fractal Dimension" as applied to the texture map generated by the Midpoint Replacement technique.
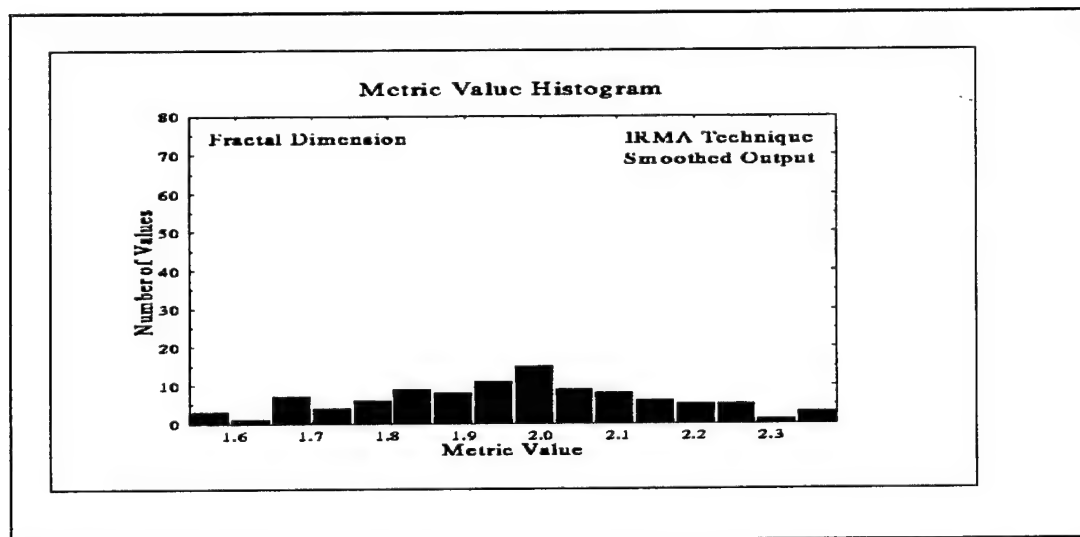
Figure D-24. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "Fractal Dimension" as applied to the texture map generated by the IRMA (Smoothed Output) technique.
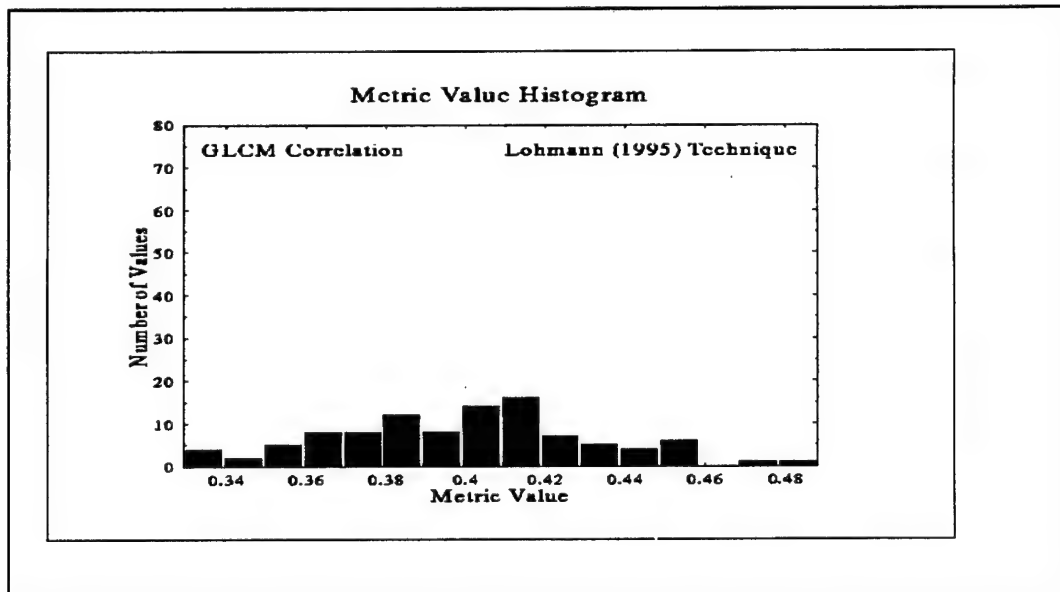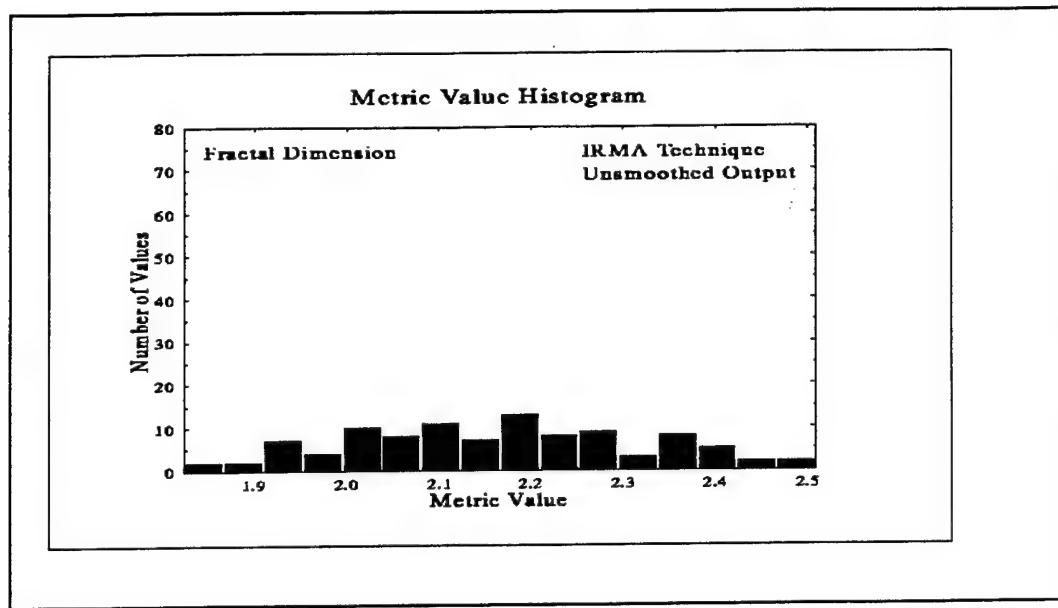
Figure D-25. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "Fractal Dimension" as applied to the texture map generated by the IRMA (Unsmooothed Output) technique.
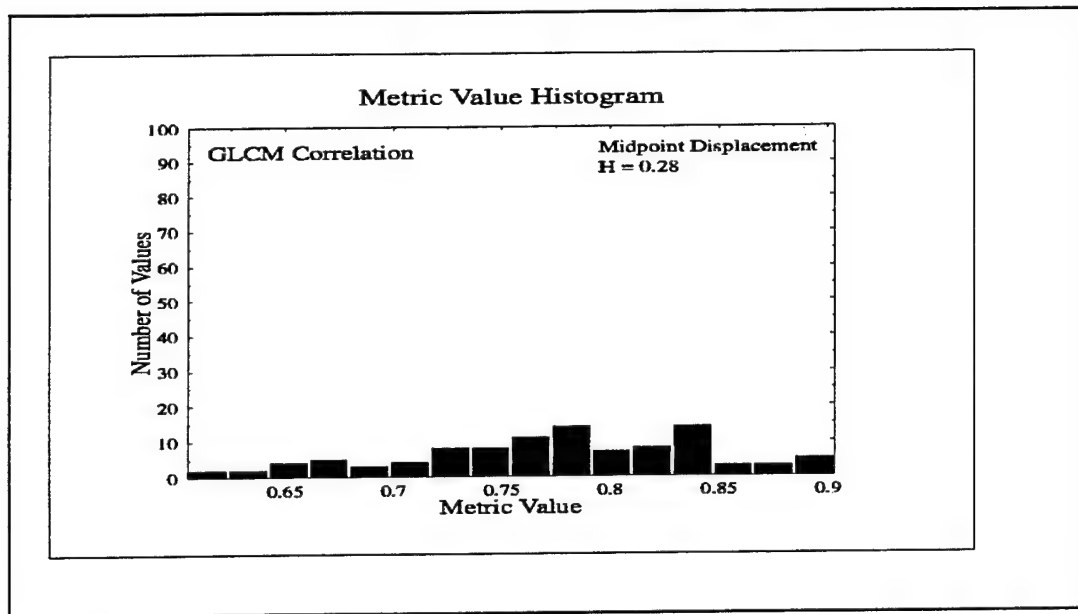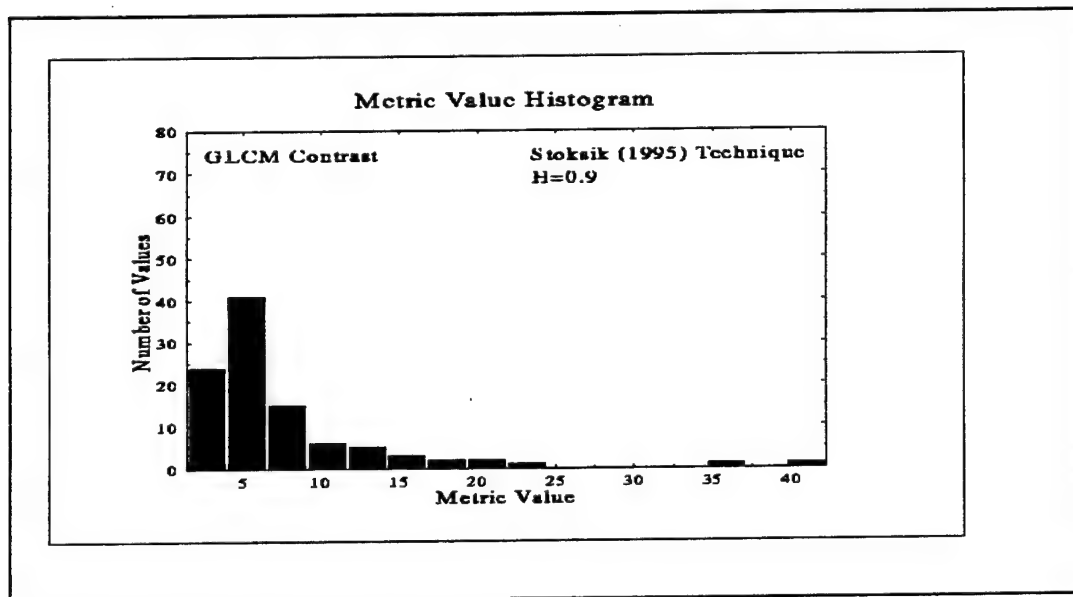

Figure D-26. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Contrast" as applied to the texture map generated by the Stoksik (1995) technique.

96
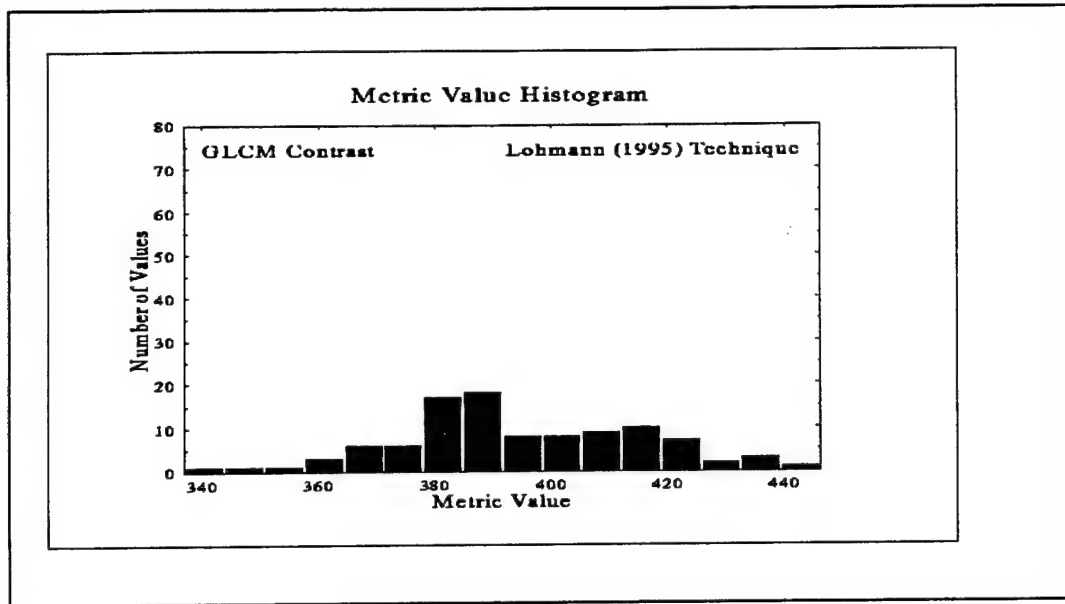
Figure D-27. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Contrast" as applied to the texture map generated by the Lohmann (1995) technique.
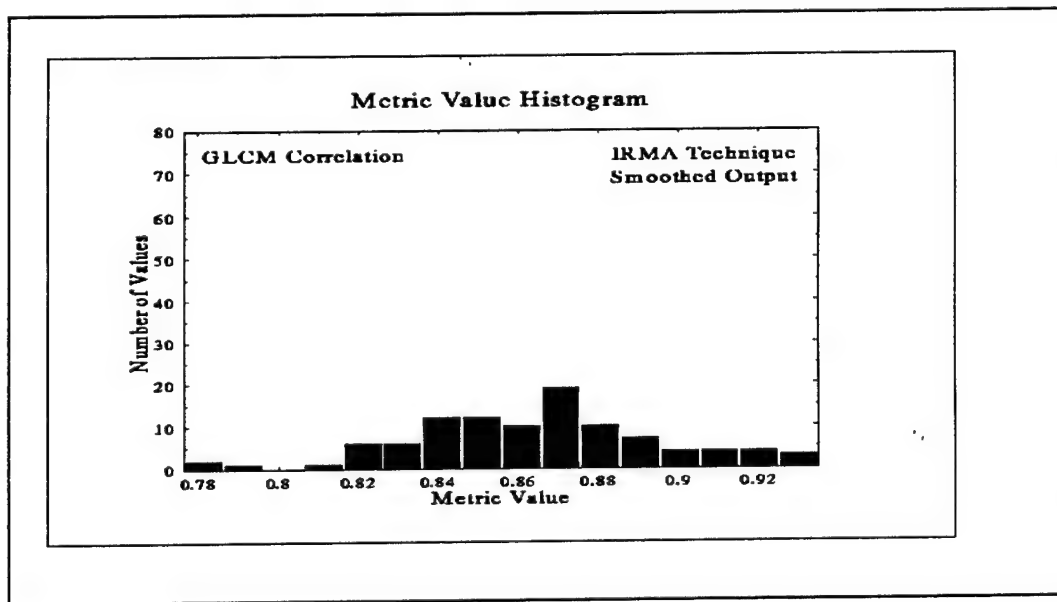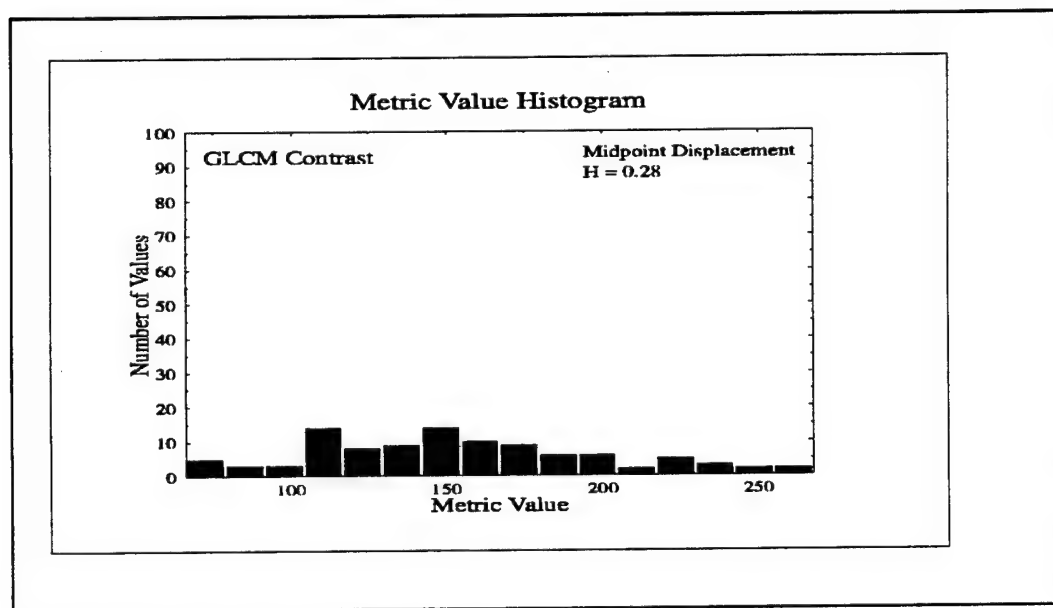

Figure D-28. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Contrast" as applied to the texture map generated by the Midpoint Replacement technique.
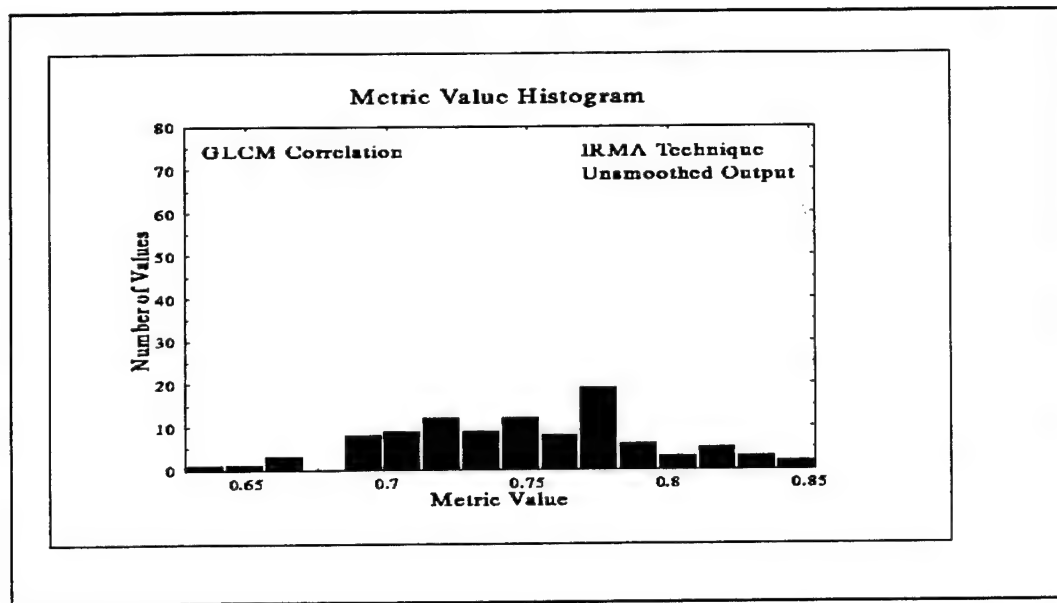
97
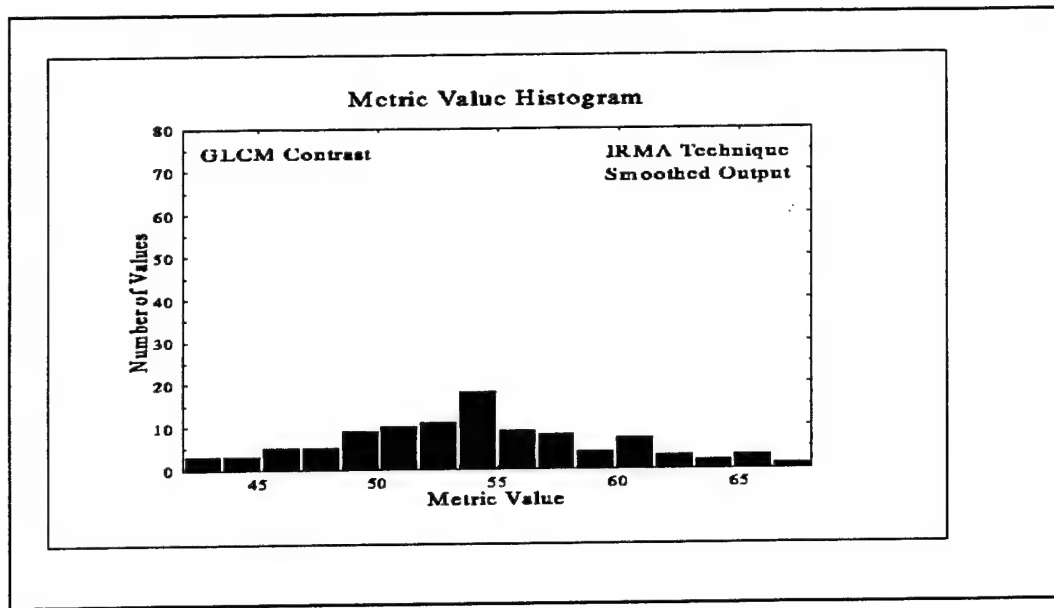
Figure D-29. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Contrast" as applied to the texture map generated by the IRMA (Smoothed Output) technique.
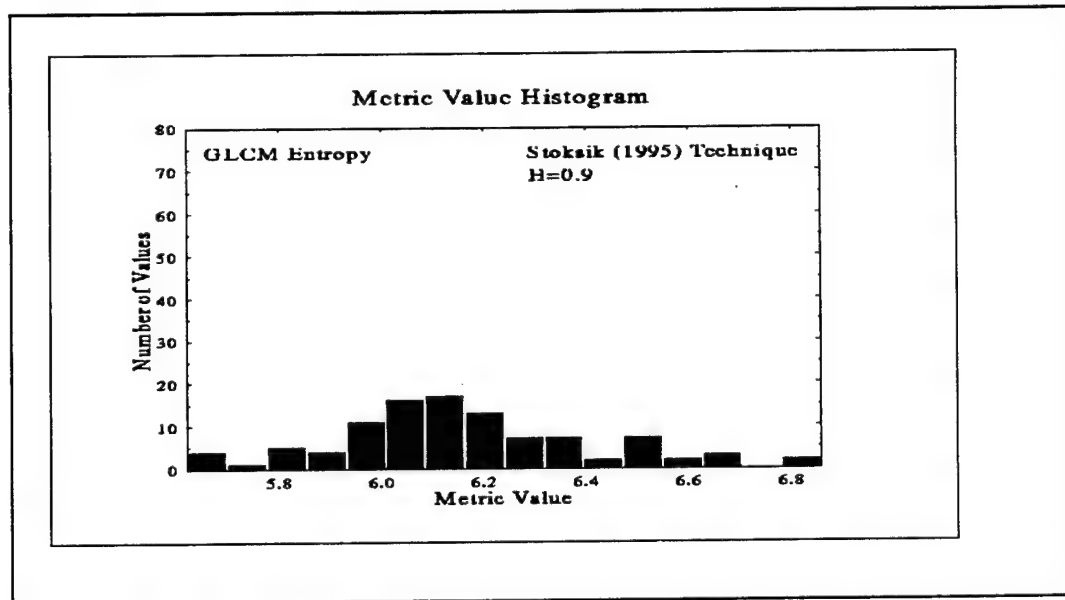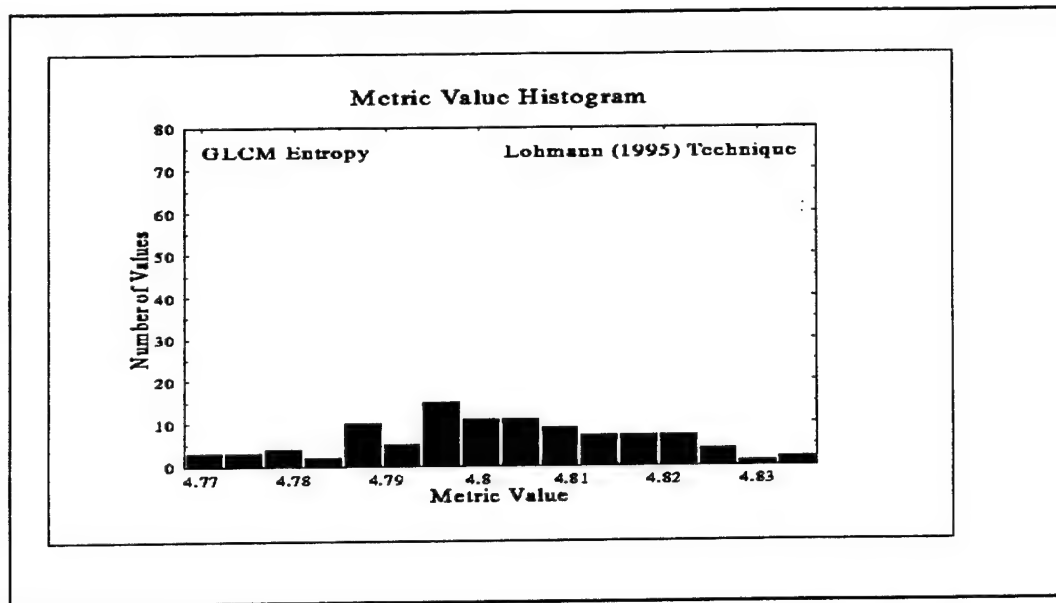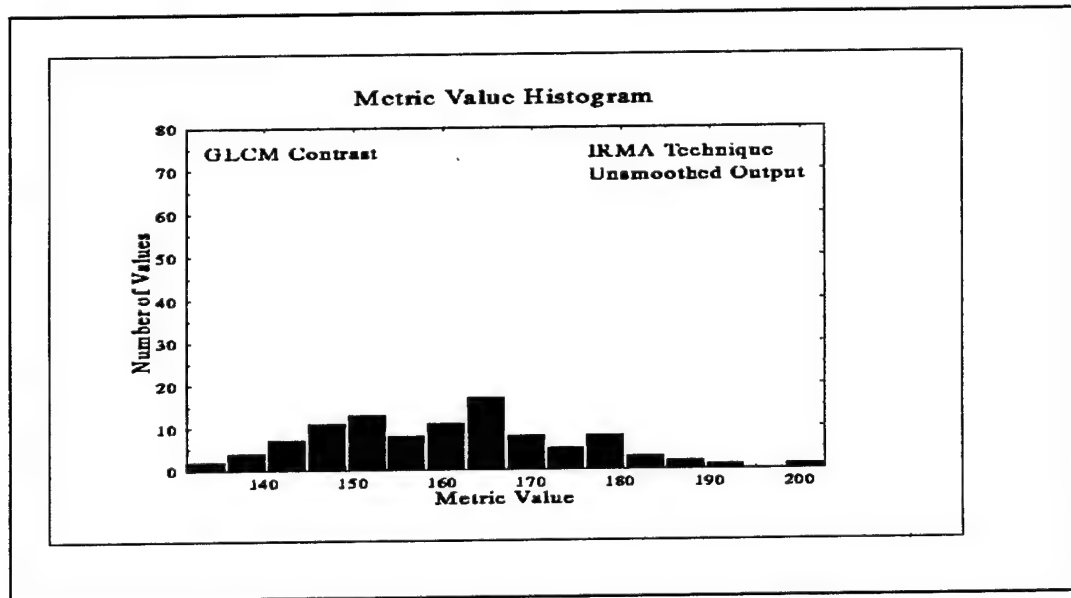

Figure D-30. One of the histograms used to derive the parameters listed in Table 3 of the main body of the report. This particular histogram shows the distribution of the metric "GLCM Contrast" as applied to the texture map generated by the IRMA (Unsmoothed Output) technique.

98

References: Appendix D

Lohmann, G., "Analysis and Synthesis of Textures: A Co-Occurrence-Based Approach," *Comput. & Graphics*, **19**, 1, p 29-36, 1995.

Stoksik, M.A., R.G. Lane, and D.T. Nguyen, "Practical Synthesis of Accurate Fractal Image," *Graphical Models and Image Processing*, **57**, 3, p 206-219, 1995.

Appendix E

AR (1) Process: A Brief Discussion

According to Kendall (1976), an autoregressive process is a stationary series which is "...generated by a mechanism in which the value of the series at time t is expressed in terms of the past values -- a systematic dependence on past history -- plus a 'disturbance' term ε happening at time t." Stationarity is defined here as applying to a series which has had its trend removed or has never had a trend. The autoregressive process of order p is one defined by :

$$u_t = -\alpha_1 u_{t-1} - \alpha_2 u_{t-2} - \cdots - \alpha_p u_{t-p} + \varepsilon_t .$$

Because the form of this equation suggests a regression process, the term "autoregressive" has been coined -- in fact, this is not necessarily so. The first order autoregressive process, AR(1), is also know as a Markov process and is defined by:

$$u_t = -\alpha_1 u_{t-1} + \varepsilon_t .$$

Rewriting this (still following Kendall) as:

$$u_t = \rho u_{t-1} + \varepsilon_t$$

then allows, after rearranging of terms and further calculations, for the determination that the autocorrelation of u_t at lag 1 (ρ) is the same as the value of the coefficient $-\alpha_1$ in the above equation. The limits on the value of ρ are that it be less than or equal to 1 and, if the autocorrelation function is to decay without oscillation, then it should also be positive; otherwise, the autocorrelation function will oscillate as it decays.

Another way of considering the AR(1) process is that it is a linear process which has as its input white noise (Jenkins and Watts, 1969) which is filtered through the equivalent of an RC circuit whose transfer function is given by:

$$h(v) = \frac{1}{T} e^{-\frac{v}{T}} .$$

102

Such a linear process is given by:

$$T\frac{dX(t)}{dt} + (X(t) - \mu) = Z(t)$$

where:

$X(t)$ = system output
$Z(t)$ = system input (white noise)
T = system time constant
μ = mean of the process.

This has a digital analog which is just that described above and given by Kendall.

References: Appendix E

Jenkins, G.W., and D.G. Watts, *Spectral Analysis And Its Applications*, Holden-Day, San Francisco, CA, 1969.

Kendall, M., *Time-Series, Second Edition*, Hafner Press, New York, 1976.

Appendix F

Mid-Point Displacement Code

```
/***************************************************
fractscn.c implements the mid-point displacement pseudocode found on page 100 of the book "The Science of Fractal
Images" by Peitgen et al.
   It limits the greylevels output to values between 0 and 127 by setting the constant integer max_pix_val=127.
The value sigma described in the Peitgen book is set to 1. and the image output of the function MidPointFM2D is scaled
and offset to have a mean and a standard deviation specified by the command line parameters mean and stdev.
   ***************************************************/


#include <alloc.h>fs

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <time.h>

#include <mathlib.h>


float gaussian(float,float);
void MidPointFM2D( float huge*,int,float,float,int,unsigned int);
float f4(float,float,float,float);
float f3(float,float,float);


void main(int argc, char *argv[])
{
    FILE *outptr1;
    float huge *Y;
```

```c
int maxlevel;//maximal # of recursions,N=2^maxlevel
float sigma;//initial standard deviation
float H;//parameter H determines fractal dimension D=3-H
int addition;//parameter(0/!=0)turns random additions off/on)
const int max_pix_val=127;
unsigned int seed;//seed value for random number generator
unsigned char ucharxij;
float maxxij=-1000000.,minxij=1000000.;
double sum, sumsq,temp;
float tmean,tvar,tstdev;
float mean,stdev;
int i,j,N;

if(argc <6){
    printf("*****The proper command line arguments are:\n");
    printf(" fractscn.exe maxlevel mean stdev H addition seed\n");
    printf(" int     maxlevel = maximal number of recursions\n");
    printf(" float     mean = mean of target texture\n");
    printf(" float     stdev = standard deviation of target         texture\n");
    printf(" float      H = parameter determining fractal dim D= 3 - H\n");
    printf(" int     addition = parameter turning random additions on/off(!=0/0)\n");
    printf(" unsigned int seed = seed parameter-0 for time seed\n");
    printf("\n");
    printf(" Size of image = N by N where N=2^{maxlevel}\n");
    exit(0);
```

```c
        }

if( (outptr1=fopen("mpscn.pix", "w+b")) == NULL)
        {
            printf("\nCould not open output file");
            exit(0);
        }

maxlevel=atoi(argv[1]);
N=pow(2,maxlevel);

printf("Memory in far heap: %lu\n",farcoreleft());
Y=(float huge *)farcalloc(((long)N+1)*((long)N+1),sizeof(float));
printf("Memory left in far heap: %lu\n",farcoreleft());

sigma=1.;/*set sigma to 1.*/
mean=atof(argv[2]);
stdev=atof(argv[3]);
H=atof(argv[4]);
addition=atoi(argv[5]);/*false=0,true=any non-zero*/
seed = (unsigned)atoi(argv[6]);
if(seed == 0)seed = (unsigned)time(NULL);
printf("seed= %u\n",seed);

    /*call function to generate fractal image */
MidPointFM2D(Y,maxlevel,sigma,H,addition,seed);
```

```c
/*calculate max, min, mean and variance of N by N subimage*/

    sum=0;
    for(i=0;i<N;++i){
        for(j=0;j<N;++j){
            if( Y[i*(N+1)+j] < minxij ) minxij=Y[i*(N+1)+j];
            if( Y[i*(N+1)+j] > maxxij ) maxxij=Y[i*(N+1)+j];
            sum += (double)Y[i*(N+1)+j];
        }
    }

    tmean=(float)(sum/ ( (double)N*(double)N) );
    printf("%d by %d subimage maxxij= %f, minxij= %f, mean= %f\n",N,N,maxxij,minxij,tmean);
    sumsq=0;
    for(i=0;i<N;++i){
        for(j=0;j<N;++j){
            temp=(double)(Y[i*(N+1)+j]-tmean);
            sumsq += temp*temp;
        }
    }

    tvar=(float)(sumsq/( (double)N*(double)N-1. ));
    tstdev=(float)sqrt((double)tvar);

    printf("%d by %d subimage var= %f, stdev= %f\n",N,N,tvar,tstdev);

        /*output square N by N array*/
    for(i=0;i<N;++i){
```

```c
for(j=0;j<N;++j){
  Y[i*(N+1)+j] = (Y[i*(N+1)+j]-tmean)*stdev/tstdev+mean;
  if(Y[i*(N+1)+j] < 0.)Y[i*(N+1)+j]=0.;
  if(Y[i*(N+1)+j] > (float)max_pix_val )Y[i*(N+1)+j]=(float)max_pix_val;
  ucharxij=(unsigned char)Y[i*(N+1)+j];
  fwrite(&(ucharxij),1,1,outptr1);
}

}

  /*calculate max, min, mean and variance of scaled N by N subimage*/
maxxij=-1000000.;minxij=1000000.;
sum=0;
for(i=0;i<N;++i){
  for(j=0;j<N;++j){
    if( Y[i*(N+1)+j] < minxij ) minxij=Y[i*(N+1)+j];
    if( Y[i*(N+1)+j] > maxxij ) maxxij=Y[i*(N+1)+j];
    sum += (double)Y[i*(N+1)+j];
  }
}

tmean=(float)(sum/ ( (double)N*(double)N) );
printf(""%d by %d subimage(scaled) max= %f, min= %f, mean= %f\n",N,N,maxxij,minxij,tmean);
sumsq=0;
for(i=0;i<N;++i){
  for(j=0;j<N;++j){
    temp=(double)(Y[i*(N+1)+j]-tmean);
```

```c
            sumsq += temp*temp;

            }

      }

tvar=(float)(sumsq/( (double)N*(double)N-1. ));
tstdev=(float)sqrt((double)tvar);
printf("%d by %d subimage(scaled) var= %f, stdev= %f\n",N,N,tvar,tstdev);

farfree((float far *)Y);
fclose(outptr1);

}/*end main*/

void MidPointFM2D(float huge *X,int maxlevel,float sigma,float H,int addition,\
                  unsigned int seed)

{
/*    float huge *X: stores a real array of size (N+1)*(N+1)
      int maxlevel: maximal # of recursions,N=2^maxlevel
      float sigma: initial standard deviation
      float H: parameter H determines fractal dimension D=3-H
      int addition: parameter(0/!=0)turns random additions off/on)
      unsigned int seed: seed value for random number generator */

      int i,N,stage;
      float delta;//real holding standard deviation
      int x,y,y0,D,d;//integer array indexing variables
```

```c
srand(seed);
N=pow(2,maxlevel);
        /*set the initial random corners*/
delta=sigma;
printf("delta= %f N= %d\n",delta,N);
X[0]=delta*gaussian(0.,1.);
X[N]=delta*gaussian(0.,1.);
X[N*(N+1)]=delta*gaussian(0.,1.);
X[N*(N+1)+N]=delta*gaussian(0.,1.);
D=N;
d=N/2;
for(stage=1;stage<=maxlevel;++stage){
        /*going from grid type I to type II*/
delta *= pow(.5,.5*H);
        /*interpolate and offset points*/
for(x=d;x<=N-d;x+=D){
    for(y=d;y<=N-d;y+=D){
        X[x*(N+1)+y]=f4(delta,X[(x+d)*(N+1)+y+d],X[(x+d)*(N+1)+y-d],\
                        X[(x-d)*(N+1)+y+d],X[(x-d)*(N+1)+y-d]);
        }
    }
        /*displace other points also if needed*/
if(addition){
    for(x=0;x<=N;x+=D){
```

```c
        for(y=0;y<=N;y+=D){
            X[x*(N+1)+y] += delta*gaussian(0.,1.);
        }
    }
}/*end if*/
    /*going from grid type II to type I*/
delta *= pow(.5,.5*H);
    /*interpolate and offset boundary grid points*/
for(x=d;x<=N-d;x+=D){
    X[x*(N+1)] = f3(delta,X[(x+d)*(N+1)],X[(x-d)*(N+1)],X[x*(N+1)+d]);
    X[x*(N+1)+N] = f3(delta,X[x+d*(N+1)+N],X[(x-d)*(N+1)+N],X[x*(N+1)+N-d]);
    X[x] = f3(delta,X[x+d],X[x-d],X[d*(N+1)+x]);
    X[N*(N+1)+x] = f3(delta,X[N*(N+1)+x+d],X[N*(N+1)+x-d],X[(N-d)*(N+1)+x]);
}/*end x for */
    /*interpolate and offset interior grid points*/
for(x=d;x<=N-d;x+=D){
    for(y=D;y<=N-d;y+=D){
        X[x*(N+1)+y] = f4(delta,X[x*(N+1)+y+d],X[x*(N+1)+y-d],\
                        X[(x+d)*(N+1)+y],X[(x-d)*(N+1)+y]);
    }
}/*end x for*/
for(x=D;x<=N-d;x+=D){
    for(y=d;y<=N-d;y+=D){
        X[x*(N+1)+y] = f4(delta,X[x*(N+1)+y+d],X[x*(N+1)+y-d],\
                        X[(x+d)*(N+1)+y],X[(x-d)*(N+1)+y]);
    }
}
```

```
            }

        }/*end x for*/
            /*displace other points also if needed*/
        if(addition){
            for(x=0;x<=N;x+=D){
                for(y=0;y<=N;y+=D){
                    X[x*(N+1)+y] += delta*gaussian(0.,1.);
                }
            }
            for(x=d;x<=N-d;x+=D){
                for(y=d;y<=N-d;y+=D){
                    X[x*(N+1)+y] += delta*gaussian(0.,1.);
                }
            }
        }/*end if*/

        D /=2;
        d /=2;
    }/*end stage for loop*/
}/*end MidPointFM2D*/

float gaussian(float mean,float stdev)
//Initialize rand by srand before this function is called.
{
    float u,v,x;
```

```
double pi=3.14159265358979;
u=(float)((float)rand()/(float)RAND_MAX);
if(u==0.)u=.000000001;
v=(float)((float)rand()/(float)RAND_MAX);
if(v==0.)v=.000000001;
x=(float)(sqrt(-2.*log((double)u))*cos(2.*pi*(double)v)*stdev)+mean;
return x;
}

float f4(float delta,float x0,float x1,float x2,float x3){
return((x0+x1+x2+x3)/4.+delta*gaussian(0.,1.));
}

float f3(float delta,float x0,float x1,float x2){
return((x0+x1+x2)/3.+delta*gaussian(0.,1.));
}
```

Appendix G

GLCM Code

```
/*********************************************************************
glcmtex.c implements the algorithm described in the Gabriele Lohmann article
"Analysis and Synthesis of Textures: A Co-Occurrence-Based Approach".
This code implements the "one sided" GLCM for 0,45,90,135 degrees.
The number of greylevels is fixed at 128.
The rows and columns of the input and output images are fixed at 32.
The value of the initial seed for the random number generator is fixed at 7 in
the function txtglcm1.
*********************************************************************/

#include <alloc.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <mathlib.h>

void in_image(FILE*,int,int,int huge*);
void in_image_char(FILE *,int,int,int huge *);
void glcm(int huge*,int huge*,int,int,int,int);
void rowsums(int huge *,int huge *,int);
void corner_br(int huge *,int huge *,int,int,int);
void find_hist(int huge *,int huge *,int huge *,int huge *,int huge *,int);
void ran_image(int huge *,int huge *,int,int,int);
long int metric(int huge *,int huge *,int);
```

```c
long int change(int huge *,int huge *,int huge *,int,int,int,int,int,int,int,int,int);
void update_glcm(int huge *,int huge *,int huge *,int,int,int,int,int,int,int,int,int);
void out_mess(char [],FILE *,int,int,int huge *);
void sout_mess(char [],FILE *,int,int,int huge *);
void out_image_char(FILE *,int,int,int huge *);
int in_bounds(int,int,int,int);
void swap(int huge *,int,int,int,int,int,int);
void pick2(int,int,int *,int *,int *,int *);
void check_add(int long [8],int [8],int *,int long,int);
void txtglcm1(int,int,int huge *,int huge *,int,int,int huge *,int huge *,int huge *,\
    int huge *,int huge *,int huge *,int huge *,int huge *,int long *,double);
FILE *outptr1, *infptr1,*outptr2;


void main(int argc, char *argv[])

{
    int huge *target;
    int huge *h0,huge *h45,huge *h90,huge *h135;
    int huge *t0,huge *t45,huge *t90,huge *t135;
    int huge *texture,huge * hist;
    int i,j,r,c,rows,cols,greylevels,enough=0,i1,i2,j1,j2;
    int long *delta;
    double T;


        /* the number of rows and columns are fixed at 32. */
```

```c
rows=32;
cols=32;

/*greylevels set to 128 to limit the size of the huge arrays h0,h45,etc.
any increase in greylevels must be matched by increase in array sizes
allocated by the farcalloc statements below */

greylevels=128;

/*set the initial annealing temperature*/
T=100.*(double)rows*(double)cols;

outptr1=fopen("out", "wt");
outptr2=fopen("texture","w+b");
if((infptr1=fopen("target","rb"))==NULL){
    printf("\nCould not open input file 1\n");
    fclose(infptr1);
    exit(0);

}

/*the array sizes allocated here assume greylevels <= 128 */
printf("\nMemory left on the far heap:   %lu\n",farcoreleft());
target=(int huge *)farcalloc(16384UL,(unsigned long)sizeof(int));
h0=(int huge *)farcalloc(16384UL,(unsigned long)sizeof(int));
h45=(int huge *)farcalloc(16384UL,(unsigned long)sizeof(int));
h90=(int huge *)farcalloc(16384UL,(unsigned long)sizeof(int));
```

```c
h135=(int huge *)farcalloc(16384UL,(unsigned long)sizeof(int));
t0=(int huge *)farcalloc(16384UL,(unsigned long)sizeof(int));
t45=(int huge *)farcalloc(16384UL,(unsigned long)sizeof(int));
t90=(int huge *)farcalloc(16384UL,(unsigned long)sizeof(int));
t135=(int huge *)farcalloc(16384UL,(unsigned long)sizeof(int));
hist=(int huge *)farcalloc(256UL,(unsigned long)sizeof(int));
texture=(int huge *)farcalloc(16384UL,(unsigned long)sizeof(int));
printf("\nMemory left on the far heap:   %lu\n",farcoreleft());

     /*read in a target image and output it to screen*/
in_image_char(infptr1,rows,cols,target);
out_mess("target image",outptr1,rows,cols,target);

     /*call txtglcm1 function to generate a texture*/
txtglcm1(rows,cols,target,texture,greylevels,\
         h0,h45,h90,h135,t0,t45,t90,t135,hist,&delta,T);


farfree((int far *)target);
farfree((int far *)h0);
farfree((int far *)h45);
farfree((int far *)h90);
farfree((int far *)h135);
farfree((int far *)t0);
farfree((int far *)t45);
farfree((int far *)t90);
```

```
        farfree((int far *)t135);
        farfree((int far *)hist);
        farfree((int far *)texture);
        fclose(infptr1);
        fclose(outptr1);

}/*end main*/

void txtglcm1(int rows,int cols,int huge *target,int huge *texture,int greylevels,\
      int huge *h0,int huge *h45,int huge *h90,int huge *h135,int huge *t0,\
      int huge *t45,int huge *t90,int huge *t135,int huge *hist,int long *delta,double T){

/*    This function generates a texture based on the four GLCMs of the "target" image which has "rows" rows and "cols"
columns.

      The output texture is called "texture". It has similar dimensions.
      The matrices h0,h45,h90,h135,t0,t45,t90,t135 are GLCM matrices of dimension "greylevels" by "greylevels" in size.
      "hist" is an array containing the histogram of the greylevels in "target".
      "delta" is a value measuring the difference in the GLCMs of the "target" image and the generated "texture" image. It is
based on the Manhattan metric.

      The initial seed value for the random number generator is fixed at 7

*/

      int rs0[256],rs45[256],rs90[256],rs135[256];
      int br[256];
      int old_delta0,old_delta45,old_delta90,old_delta135;
```

122

```
int long inc,inc0,inc45,inc90,inc135,swap_cnt_inc,swap_cnt_u,cnt;
float u,toss,percent,percentu;
int i,j,enough=0,i1,i2,j1,j2;
unsigned int seed;

        /*find h0,h45,h90,h135, the GLCMs of "target"*/
glcm(target,h0,greylevels,rows,cols,0,1);
glcm(target,h45,greylevels,rows,cols,1,1);
glcm(target,h90,greylevels,rows,cols,1,0);
glcm(target,h135,greylevels,rows,cols,1,-1);

        /*find histogram of "target" by first calculating the rowsums*/
rowsums(h0,rs0,greylevels);
rowsums(h45,rs45,greylevels);
rowsums(h90,rs90,greylevels);
rowsums(h135,rs135,greylevels);

        /*find the lower right corner value*/
corner_br(br,target,greylevels,rows,cols);

        /*find histogram*/
find_hist(hist,rs0,rs45,rs90,br,greylevels);

        /*find initial random image "texture" with this histogram*/
seed=7;
```

```
srand(seed);
ran_image(hist,texture,greylevels,rows,cols);
out_mess("random texture start",outptr1,rows,cols,texture);

    /*find t0,t45,t90,t135 the GLCMs of "texture"*/
glcm(texture,t0,greylevels,rows,cols,0,1);
glcm(texture,t45,greylevels,rows,cols,1,1); .
glcm(texture,t90,greylevels,rows,cols,1,0);
glcm(texture,t135,greylevels,rows,cols,1,-1);

    /*find taxicab distance between target glcm's and texture glcm's*/
old_delta0   = metric(h0,t0,greylevels);
old_delta45  = metric(h45,t45,greylevels);
old_delta90  = metric(h90,t90,greylevels);
old_delta135 = metric(h135,t135,greylevels);
*delta=old_delta0+old_delta45+old_delta90+old_delta135;
printf("starting delta= %ld\n",*delta);
fprintf(outptr1,"starting delta= %ld\n",*delta);


do{

    cnt=0;
    swap_cnt_inc=0;
    swap_cnt_u=0;
```

```c
for(i=0;i<rows*cols;++i){
        /*pick two distinct pixels to swap*/
    pick2(rows,cols,&i1,&j1,&i2,&j2);
        /*find inc, the change in the metric caused by the swap */
    inc0=change(texture,h0,t0,greylevels,rows,cols,i1,j1,i2,j2,0,1);
    inc45=change(texture,h45,t45,greylevels,rows,cols,i1,j1,i2,j2,1,1);
    inc90=change(texture,h90,t90,greylevels,rows,cols,i1,j1,i2,j2,1,0);
    inc135=change(texture,h135,t135,greylevels,rows,cols,i1,j1,i2,j2,1,-1);
    inc=inc0+inc45+inc90+inc135;
    if(inc<0){ /*if there is an improvement we swap the pixels*/
        /*call to update_glcm must preceed swap of pixels*/
        update_glcm(texture,t0,greylevels,rows,cols,i1,j1,i2,j2,0,1);
        update_glcm(texture,t45,greylevels,rows,cols,i1,j1,i2,j2,1,1);
        update_glcm(texture,t90,greylevels,rows,cols,i1,j1,i2,j2,1,0);
        update_glcm(texture,t135,greylevels,rows,cols,i1,j1,i2,j2,1,-1);
        swap(texture,cols,rows,i1,j1,i2,j2);
        *delta += inc;
        ++swap_cnt_inc;
    }else{
        u=(float)(1./(1.+exp((double)(*delta+inc)/T)));
        toss=(float)rand()/(float)RAND_MAX;
        if(toss<u){ /*this is the simulated annealing part. We swap even though
                delta increases as a result*/
            /*call to update_glcm must preceed swap of pixels*/
            update_glcm(texture,t0,greylevels,rows,cols,i1,j1,i2,j2,0,1);
```

```
                update_glcm(texture,t45,greylevels,rows,cols,i1,j1,i2,j2,1,1);
                update_glcm(texture,t90,greylevels,rows,cols,i1,j1,i2,j2,1,0);
                update_glcm(texture,t135,greylevels,rows,cols,i1,j1,i2,j2,1,-1);
                swap(texture,cols,rows,i1,j1,i2,j2);
                *delta += inc;
                ++swap_cnt_u;
            }
        }
        ++cnt;
        if(*delta==0)break;
    }/*end for loop with index i*/
    percent=((float)(swap_cnt_inc)/(float)cnt)*(float)100.;
    percentu=((float)(swap_cnt_u)/(float)cnt)*(float)100.;
    printf("pct,pctu,del,T=%3.0f %3.0f %ld %lf\n",percent,percentu,*delta,T);
    if(percent<1.||delta==0){
        printf("best delta=%ld\n",*delta);
        fprintf(outptr1,"delta=%ld\n",*delta);
        out_mess("\"best\" texture",outptr1,rows,cols,texture);
        out_mess("target",outptr1,rows,cols,target);
        out_image_char(outptr2,rows,cols,texture);
        enough=1;
    }
    T *= .99;
}while(enough==0); /*end do while loop*/
}/*end txtglcm1 function*/
```

```c
void out_mess(char message[80],FILE *outptr,int rows,int cols,int huge *target){

/*outputs a message and the matrix "target" in ASCII format to both screen and a file */

    int i,j;
    printf("%s\n",message);
    fprintf(outptr, "%s\n",message);
    for(i=0;i<rows;++i){
        for(j=0;j<cols;++j){
            fprintf(outptr, "%d ",target[i*rows+j]);
            printf("%d ",target[i*rows+j]);
        }
        fprintf(outptr, "\n");
        printf("\n");
    }
}

void sout_mess(char message[80],FILE *outptr,int rows,int cols,int huge *target){

/*this function outputs a message and the matrix "target" to the screen*/

    int i,j;
    printf("%s\n",message);
    for(i=0;i<rows;++i){
```

```c
		for(j=0;j<cols;++j){
			printf("%d ",target[i*rows+j]);
		}
		printf("\n");
	}
}

/*This function outputs to a file a row by col matrix of unsigned char values*/

void out_image_char(FILE *outptr,int rows,int cols,int huge *target){

	int i,j;
	unsigned char tempchar;
	for(i=0;i<rows*cols;++i){
		tempchar=(unsigned char)target[i];
		fwrite(&tempchar,1,1,outptr);
	}
}

/*This function inputs from a file a matrix in ASCII format*/

void in_image(FILE *infptr,int rows,int cols,int huge *target){

	int i,j;
	char *v;
```

```c
    for(i=0;i<rows*cols;++i){
        fscanf(infptr, "%s",v);
        target[i]=atoi(v);
    }
}
```

/*This function inputs from a file a matrix of unsigned char values*/

```c
void in_image_char(FILE *infptr,int rows,int cols,int huge *target){

    int i,j;
    unsigned char tempchar;
    for(i=0;i<rows*cols;++i){
        fread(&tempchar,1,1,infptr);
        target[i]=(int)tempchar;
    }
}
```

```c
void glcm(int huge *input,int huge *output,int greylevels,int rows,int cols,int dy,int dx){
    /* This function reads the image "input" and calculates the GLCM matrix "output"
       input is a 2 dimensional (2d) array with dimensions rows*cols
       output is a 2d array with dimensions greylevels*greylevels
       dx is the offset in the horizontal dimension
       dy is the offset in the vertical dimension
       glcm is debugged for those values of dx and dy that yield GLCMs in the 0,45,90, and 135
```

```
           degree directions
    */

    int i,j,gl,glplus,jstart,jend;
    for(i=0;i<greylevels*greylevels;++i){
        output[i]=0;
    }
    if(dx >= 0){
        jstart=0;
        jend=cols-dx;
    }else{
        jstart=-dx;
        jend=cols;
    }
    for(i=0;i<rows-dy;++i){
        for(j=jstart;j<jend;++j){
            gl=i*rows+j;
            glplus=(i+dy)*rows+j+dx;
            ++output[input[gl]*greylevels+input[glplus]];
        }
    }
}

void rowsums(int huge *input,int huge *output,int greylevels){
/*   input is a 2d square matrix of size greylevels*greylevels
     output is a vector of size greylevels containing the row sums of input
```

```c
*/
    int i,j;
    for(i=0;i<greylevels;++i){
        output[i]=0;
    }
    for(i=0;i<greylevels;++i){
        for(j=0;j<greylevels;++j){
            output[i] += input[i*greylevels+j];
        }
    }
}

void find_hist(int huge *hist,int huge *rs0,int huge *rs45,int huge *rs90,int huge *br,int greylevels){

/*    This function calculates the histogram of greylevels values in "target" (see txtglcm())
      These calculations assume that the one directional GLCMs in the 0,45,90, and 135 degree directions have been calculated
      and the row sums rs0,rs45,rs90,rs135 and the bottom corner array have been found.
*/
    int i;
    for(i=0;i<greylevels;++i){
        hist[i]=(rs0[i]+rs90[i]-rs45[i]+br[i]);
    }
}

void ran_image(int huge *hist,int huge *texture,int greylevels,int rows,int cols){
```

131

```
/*   This function generates a random rows by cols image with greylevel histogram "hist".
     The random number initialization function srand() has already been called

*/

     int i,pixelval,dummy;
     for(i=0;i<rows*cols;++i){
         do{
             pixelval=rand() % greylevels;
             if( hist[pixelval] > 0 )break;

         }
         while(1);
         texture[i]=pixelval;
         --hist[pixelval];

     }

}

long int metric(int huge *m1,int huge *m2,int greylevels){

/*Measures the Manhattan or taxicab distance between the matrices "m1" and "m2".
*/

     int i;
     long int delta;
     delta=0;
```

```c
   for(i=0;i<greylevels*greylevels;++i){
       delta += (int long)abs(m1[i]-m2[i]);
   }
   return(delta);

}
```

/*   This function calculates the bottom corner occupancy array for the image "target".
   For example if greylevels={0,1,2,3} and the greylevel in the bottom right corner of "target"
   is a 3, this function returns the array {0,0,0,1}.

*/

```c
void corner_br(int huge *br,int huge *target,int greylevels,int rows,int cols){

   int i;
   for(i=0;i<greylevels;++i){
       br[i]=0;
   }
   br[target[rows*cols-1]]=1;

}

long int change(int huge *in,int huge *h,int huge *t,int g,int rows,int cols,\
                int i1,int j1,int i2,int j2,int dy,int dx){
```

/*   This function calculates the value "delta" resulting from a swap of
   the greylevel values at pixels (j1,i1) and (j2,i2). "j1" is a row index.

"Delta" is the change in the "distance" between the target and texture GLCMs before and after the swap. So if "delta" is positive, the swap has moved the texture GLCM farther away from the target GLCM.

*/

```c
int i,i1m,i1p,i2m,i2p,j1m,j1p,j2m,j2p,P,Pm,Pp,Q,Qm,Qp,a,b,c,d,e,f;
int long delta;
int long address[8]={0L,0L,0L,0L,0L,0L,0L,0L};
int deltat[8]={0,0,0,0,0,0,0,0};
int index=0;
```

/*calculate the indices of the points preceding and coming after the two pixels*/

```c
i1m=i1-dx;
j1m=j1-dy;
i1p=i1+dx;
j1p=j1+dy;
i2m=i2-dx;
j2m=j2-dy;
i2p=i2+dx;
j2p=j2+dy;
```

/*calculate the target matrix indices*/

```c
P=j1*cols+i1;
Pm=j1m*cols+i1m;
Pp=j1p*cols+i1p;
Q=j2*cols+i2;
```

```c
Qm=j2m*cols+i2m;
Qp=j2p*cols+i2p;
/*find the values occurring in the target matrix*/

a=in[Pm];
b=in[P];
c=in[Pp];
d=in[Qm];
e=in[Q];
f=in[Qp];
delta=0;
if(b==e)return(delta);

/*cover all the three cases*/
if(i2==i1m&&j2==j1m){
   if(in_bounds(i1m,j1m,cols,rows)){
      check_add(address,deltat,&index,(long)(a*g+b),-1);
      check_add(address,deltat,&index,(long)(b*g+a),1);
   }
   if(in_bounds(i1p,j1p,cols,rows)){
      check_add(address,deltat,&index,(long)(b*g+c),-1);
      check_add(address,deltat,&index,(long)(a*g+c),1);
   }
   if(in_bounds(i2m,j2m,cols,rows)){
      check_add(address,deltat,&index,(long)(d*g+a),-1);
      check_add(address,deltat,&index,(long)(d*g+b),1);
```

```
else if(i2==i1p&&j2==j1p){
    if(in_bounds(i1m,j1m,cols,rows)){
        check_add(address,deltat,&index,(long)(a*g+b),-1);
        check_add(address,deltat,&index,(long)(a*g+c),1);
    }
    if(in_bounds(i1p,j1p,cols,rows)){
        check_add(address,deltat,&index,(long)(b*g+c),-1);
        check_add(address,deltat,&index,(long)(c*g+b),1);
    }
    if(in_bounds(i2p,j2p,cols,rows)){
        check_add(address,deltat,&index,(long)(e*g+f),-1);
        check_add(address,deltat,&index,(long)(d*g+f),1);
    }
}
else{
    if(in_bounds(i1m,j1m,cols,rows)){
        check_add(address,deltat,&index,(long)(a*g+b),-1);
        check_add(address,deltat,&index,(long)(a*g+e),1);
    }
    if(in_bounds(i1p,j1p,cols,rows)){
        check_add(address,deltat,&index,(long)(b*g+c),-1);
        check_add(address,deltat,&index,(long)(e*g+c),1);
    }
```

```c
        if(in_bounds(i2m,j2m,cols,rows)){
            check_add(address,deltat,&index,(long)(d*g+e),-1);
            check_add(address,deltat,&index,(long)(d*g+b),1);
        }
        if(in_bounds(i2p,j2p,cols,rows)){
            check_add(address,deltat,&index,(long)(e*g+f),-1);
            check_add(address,deltat,&index,(long)(b*g+f),1);
        }
    }
    /*accumulate the delta*/
    for(i=0;i<8;++i){
        delta += (long)(abs(h[address[i]]-t[address[i]]-deltat[i])\
                -abs(h[address[i]]-t[address[i]]));
    }
    return(delta);
}

void update_glcm(int huge *in,int huge *t,int g,int rows,int cols,\
                int i1,int j1,int i2,int j2,int dy,int dx){

/*  This function updates the GLCM matrix resulting from the swap of the two pixels
    specified by (j1,i1) and (j2,i2).
*/

int i1m,i1p,i2m,i2p,j1m,j1p,j2m,j2p,P,Pm,Pp,Q,Qm,Qp,a,b,c,d,e,f;
```

```
int long delta;
i1m=i1-dx;
j1m=j1-dy;
i1p=i1+dx;
j1p=j1+dy;
i2m=i2-dx;
j2m=j2-dy;
i2p=i2+dx;
j2p=j2+dy;
P=j1*cols+i1;
Pm=j1m*cols+i1m;
Pp=j1p*cols+i1p;
Q=j2*cols+i2;
Qm=j2m*cols+i2m;
Qp=j2p*cols+i2p;
a=in[Pm];
b=in[P];
c=in[Pp];
d=in[Qm];
e=in[Q];
f=in[Qp];
if(b==e)return;
/*cover all the three cases*/
if(i2==i1m&&j2==j1m){
  if(in_bounds(i1m,j1m,cols,rows)){
```

```
			--t[a*g+b];
			++t[b*g+a];
		}
		if(in_bounds(i1p,j1p,cols,rows)){
			--t[b*g+c];
			++t[a*g+c];
		}
		if(in_bounds(i2m,j2m,cols,rows)){
			--t[d*g+a];
			++t[d*g+b];
		}
	}
	else if(i2==i1p&&j2==j1p){
		if(in_bounds(i1m,j1m,cols,rows)){
			--t[a*g+b];
			++t[a*g+c];
		}
		if(in_bounds(i1p,j1p,cols,rows)){
			--t[b*g+c];
			++t[c*g+b];
		}
		if(in_bounds(i2p,j2p,cols,rows)){
			--t[e*g+f];
			++t[d*g+f];
		}
	}
```

```c
    }
    else{
        if(in_bounds(i1m,j1m,cols,rows)){
            --t[a*g+b];
            ++t[a*g+e];
        }
        if(in_bounds(i1p,j1p,cols,rows)){
            --t[b*g+c];
            ++t[e*g+c];
        }
        if(in_bounds(i2m,j2m,cols,rows)){
            --t[d*g+e];
            ++t[d*g+b];
        }
        if(in_bounds(i2p,j2p,cols,rows)){
            --t[e*g+f];
            ++t[b*g+f];
        }
    }
    return;
}

int in_bounds(int i,int j,int cols,int rows){

/*  This function determines whether the pixel with calculated coordinates
```

```
     (j,i) is in the image
*/

if(i<0 || i>=cols || j<0 || j>=rows)return(0);
    return(1);

}

void swap(int huge *in,int cols,int rows,int i1,int j1,int i2,int j2){

/*   This function calculates the new image resulting from the swap of two pixels
*/

    int i;
    if(i1<0 || i1>=cols)return;
    if(i2<0 || i2>=cols)return;
    if(j1<0 || j1>=rows)return;
    if(j2<0 || j2>=rows)return;
    i=in[j1*cols+i1];
    in[j1*cols+i1]=in[j2*cols+i2];
    in[j2*cols+i2]=i;

}

void pick2(int rows,int cols,int *i1,int *j1,int *i2,int *j2){

/*   This function picks two pixels at random */
```

```c
        int P,Q;
        do{
            *i1=rand() % cols;
            *j1=rand() % rows;
            *i2=rand() % cols;
            *j2=rand() % rows;
            P=*j1*cols+*i1;
            Q=*j2*cols+*i2;
        }while(P==Q);/*make sure P and Q are not the same point*/
}

void check_add(int long address[8],int deltat[8],int *index,int long address_item,int change){
    /*this fn accumulates changes to a GLCM and saves the locations(addresses)
      where the changes occur*/

    int i;
    if(*index<2){/*the first 2 items are always distinct*/
        address[*index]=address_item;
        deltat[*index]=change;
        ++*index;
    }else{
        /*check whether address_item has occurred before in the address list*/
        for(i=0;i<*index;++i){
            if(address[i]==address_item){
                deltat[i] += change;
```

```
        return;
    }

}
address[*index]=address_item;
deltat[*index]=change;
++*index;

}

}
```

Appendix H

Correlation Length Code (IRMA)

```
/*
          This C code implements the SWOE texture generation code based on the IRMA model.
  This code is a translation of the Pascal code in SWOE Report 91-3, "Generating Textures    for Synthetic  Thermal
  Scenes", by B.M. Sabol and L.K. Balick.
  The pixel greylevel values are limited to values between 0 and 127 by setting the       constant integer max_pix_val =
  127.

          The following command line parameters with a sample set are needed:
                                    rows cols stdev   mean    horiz_corr_length

                 irmatex.exe  32   32   18.5301 60.47619  3.538

          The initial random number seed, idum, a negative integer, was fixed at -7

*/

#include <alloc.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>


void in_image(FILE*,int,int,int huge*);
void out_mess(char [],FILE *,int,int,int huge *);
void sout_mess(char [],FILE *,int,int,int huge *);
float ran1();
float gasdev();
void autoregression(int huge *,int,int,float,float,float,float);
void out_char(FILE *,int,int,int huge *);
```

```c
void moving_average(int huge *,int,int);
int round(float);

FILE *outptr1,*outptr2,*outptr3;
float glr[97];
long int glix1,glix2,glix3;
int idum;/*initial negative random integer seed for ran1*/
const int max_pix_val=127;

void main(int argc, char *argv[])
{

    int huge *h0;
    int huge *texture;
    float std,mean,hcl,vcl;
    int i,j,r,c,rows,cols,i1,i2,j1,j2;
    int long *delta;

    outptr1=fopen("out", "wt");
    outptr2=fopen("outchar", "w+b");
    outptr3=fopen("smoothed", "w+b");

    printf("\nMemory left on the far heap:    %lu\n",farcoreleft());
    h0=(int huge *)farcalloc(16384UL,(unsigned long)sizeof(int));
    texture=(int huge *)farcalloc(16384UL,(unsigned long)sizeof(int));
    printf("\nMemory left on the far heap:    %lu\n",farcoreleft());
```

147

```
idum=-7;
rows=atoi(argv[1]);
cols=atoi(argv[2]);
std=atof(argv[3]);
mean=atof(argv[4]);
hcl=atof(argv[5]);
vcl=hcl;/*force isotropic texturing*/

        /*generate IRMA texture*/
autoregression(texture,rows,cols,std,mean,hcl,vcl);
out_mess("texture image",outptr1,rows,cols,texture);
out_char(outptr2,rows,cols,texture);

        /*average the IRMA texture to get a smoothed IRMA texture*/
moving_average(texture,rows,cols);
out_mess("smoothed texture image",outptr1,rows,cols,texture);
out_char(outptr3,rows,cols,texture);

farfree((int far *)h0);
farfree((int far *)texture);
fclose(outptr1);
fclose(outptr2);
fclose(outptr3);
```

```c
}/*end main*/

void moving_average(int huge *texture,int rows,int cols){
    int one_ninth,lph,lpv,i,j,tt;
    static int hold1[258];
    static int hold2[258];
    float wt_ninth;
    float mask[9]={2./9.,0.0,2./9.,0.0,1./9.,0.0,2./9.,0.0,2./9.};

    one_ninth=0.0;
    for(lph=1;lph<rows-1;++lph){
        for(lpv=1;lpv<cols-1;++lpv){
            for(i=0;i<3;++i){
                for(j=0;j<3;++j){

wt_ninth=mask[i*3+j]*(float)texture[(lph+i-1)*cols+lpv+j-1];
            one_ninth=round((float)one_ninth+wt_ninth);
                }
            }
            if(one_ninth>max_pix_val)one_ninth=max_pix_val;
            if(one_ninth<1)one_ninth=1;
            hold1[lpv]=one_ninth;
            one_ninth=0.0;
        }/*end lpv=0,etc*/
```

```c
        if(lph>1){
                for(tt=1;tt<cols-1;++tt){
                        texture[(lph-1)*cols+tt]=hold2[tt];
                }
        }

        for(tt=1;tt<cols-1;++tt){
                hold2[tt]=hold1[tt];
        }

}/*end lph=1,etc*/
        /*need to recover last row processed*/
for(tt=1;tt<cols-1;++tt){
        texture[(lph-1)*cols+tt]=hold2[tt];
}

}/*end moving_average*/

int round(float x){

        /*round(.5)=1, round(-.5)=-1*/

        return((int)floor((double)x+.5));

}

void autoregression(int huge *texture,int rows,int cols,\

        float std,float mean,float hcl,float vcl){
```

```
int i,j,h;
float rij,rijeq,zerozero,adis1,adis2,rmean,t1,t2,rstd;
float pixelprev,pixelabov,pixelupbak,adjust;
float parr[2][442];

rij=gasdev();
zerozero=rij*std+mean;
adis1=exp(-1./hcl);
adis2=exp(-1./vcl);
rmean=1.-adis1-adis2+adis1*adis2;
rmean *= mean;
t1=adis1*adis1;
t2=adis2*adis2;
rstd=sqrt(std*std*(1.-t1-t2+t1*t2));
parr[0][0]=zerozero;
for(i=1;i<=rows;++i){
    for(j=1;j<=cols;++j){
        rij=gasdev();
        rijeq=rij*rstd+rmean;
        if(i==1)parr[0][j]=zerozero;
        if(j==1)parr[1][0]=zerozero;
        pixelprev=parr[1][j-1];
        pixelabov=parr[0][j];
        pixelupbak=parr[0][j-1];
        adjust=adis1*pixelprev+adis2*pixelabov\
```

```c
                    -adis1*adis2*pixelupbak+rijeq;
            parr[1][j]=adjust;
            if(adjust<(float)1.)adjust=1.;
            if(adjust>(float)max_pix_val)adjust=max_pix_val;
            texture[(i-1)*cols+j-1] = (int)adjust;
        }
        for(h=1;h<=cols;++h) parr[0][h]=parr[1][h];
    }
}

float gasdev(){
    float fac,r,v1,v2,gasdev;
    static float glgset;
    static int gliset=0;

    if(gliset==0){
        do{
            v1=2.*ran1()-1.;
            v2=2.*ran1()-1.;
            r=v1 * v1 + v2 * v2;
        }while( r >= 1 );
        fac=sqrt(-2.*log(r)/r);
        glgset=v1*fac;
        gasdev=v2*fac;
        gliset=1;
```

```
            }else{
                   gasdev=glgset;
                   gliset=0;
           }
           return(gasdev);
   }

float ran1(){
           const long int m1=259200;
           const long int ia1=7141;
           const long int ic1=54773;
           const long int m2=124456;/* 1344567 */
           const long int ia2=8121;
           const long int ic2=28411;
           const long int m3=243000;
           const long int ia3=4561;
           const long int ic3=51349;
           const double rm1=3.850247e-6;
           const double rm2=7.4373773e-6;
           int b;
           float ran1;
           if(idum<0){
                   glix1=(ic1-idum)%m1;
                   glix1=(ia1*glix1+ic1)%m1;
                   glix2=glix1 % m2;
```

```c
        glix1=(ia1*glix1+ic1) % m1;
        glix3=glix1 % m3;
        for(b=0;b < 97;++b){
                glix1=(ia1*glix1+ic1)%m1;
                glix2=(ia2*glix2+ic2) % m2;
                glr[b]=(glix1+glix2*rm2)*rm1;

        }
        idum=1;

    }

    glix1=(ia1*glix1+ic1) % m1;
    glix2=(ia2*glix2+ic2) % m2;
    glix3=(ia3*glix3+ic3) % m3;

//  b=1+(97*glix3)/m3;
    b=(97*glix3)/m3;/*omit 1 because b is between 0 and 96 inclusive*/
    if(b>96||b<0){

        printf("pause in routine ran1");

    }
    ran1=glr[b];
    glr[b]=(glix1+glix2*rm2)*rm1;
    return(ran1);

}/*end of ran1*/

void out_mess(char message[80],FILE *outptr,int rows,int cols,int huge *target){

/*this function outputs ASCII to screen and a file a message and the matrix "target"*/
```

```c
   int i,j;
   printf("%s\n",message);
   fprintf(outptr1,"%s\n",message);
   for(i=0;i<rows;++i){
      for(j=0;j<cols;++j){
         fprintf(outptr, "%d ",target[i*cols+j]);
         printf("%d ",target[i*cols+j]);
      }
      fprintf(outptr,"\n");
      printf("\n");
   }
}

void sout_mess(char message[80],FILE *outptr1,int rows,int cols,int huge *target){

/*this function outputs ASCII to screen a message and the matrix "target"*/

   int i,j;
   printf("%s\n",message);
   for(i=0;i<rows;++i){
      for(j=0;j<cols;++j){
         printf("%d ",target[i*cols+j]);
      }
      printf("\n");
```

```c
void in_image(FILE *infptr,int rows,int cols,int huge *target){
```

/*This function inputs from a file a matrix in ASCII format*/

```c
    int i,j;
    char *v;
    for(i=0;i<rows*cols;++i){
        fscanf(infptr, "%s",v);
        target[i]=atoi(v);
}
```

```c
void out_char(FILE *outptr,int rows,int cols,int huge *target){
```

/*this function outputs a matrix of greylevels(characters) to a file*/

```c
    int i,j,maxt=-10000,mint=10000;
    unsigned long numpix;
    unsigned char u;

    for(i=0;i<rows;++i){ /*find max and min pixel value*/
        for(j=0;j<cols;++j){
```

```c
        numpix=i*cols+j;
        if(target[numpix]>maxt)maxt=target[numpix];
        if(target[numpix]<mint)mint=target[numpix];
    }
}

for(i=0;i<rows;++i){
    for(j=0;j<cols;++j){
        u=(unsigned char)target[i*cols+j];
        fwrite(&u,1,1,outptr);
    }
}
```

Appendix I

"Accurate Fractal" Code

/*July 5,1995,accfrac.c:This program implements the algorithm found in
the article "Practical Synthesis of Accurate Fractal Images" by
Stoksik et al. in Graphical Models and Image Processing Vol 57,
No. 3,pp206-219,1995.
It includes the functions AccFB2d and Gaussian
*/

```c
#include <alloc.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <mathlib.h>


float gaussian(float,float);
void AccFB2d( float huge*,int,float,float,unsigned int);
float f4(float,float,float,float,float);
float f2(float,float,float);

void main(int argc, char *argv[])
{
        FILE *outptr1;
        float huge *Y;
        int maxlevel;//maximal # of recursions,N=2^maxlevel
        float D_image;//width of the image
        float H;//parameter H determines fractal dimension D=3-H
```

```c
const int max_pix_val=127;
unsigned int seed;//seed value for random number generator
unsigned char ucharxij;
float maxxij=-1000000.,minxij=1000000.;

int i,j,N;

if(argc <4){

     printf("*****The proper command line arguments are as follows:\n");
     printf(" fractscn.exe maxlevel,D_image,H,seed\n");
     printf(" int    maxlevel = maximal number of recursions\n");
     printf(" float   D_image = width of the image\n");
     printf(" float      H = parameter determining fractal dim D= 3 - H\n");
     printf(" unsigned int seed = seed parameter-0 for time seed\n");
     printf(" Size of image = (N+1) by (N+1) where N=2^{maxlevel}\n");
     exit(0);

}

if( (outptr1=fopen("acc.pix","w+b")) == NULL)
{

     printf("\nCould not open output file");
     exit(0);

}

maxlevel=atoi(argv[1]);
N=pow(2,maxlevel);
printf("Memory in far heap: %lu\n",farcoreleft());
Y=(float huge *)farcalloc(((long)N+1)*((long)N+1),sizeof(float));
```

161

```c
printf("Memory left in far heap: %lu\n",farcoreleft());
D_image=atof(argv[2]);
H=atof(argv[3]);
seed = (unsigned)atoi(argv[4]);
if(seed == 0)seed = (unsigned)time(NULL);
printf("we got to here:before call to  acc\n");
AccFB2d(Y,maxlevel,D_image,H,seed);
printf("we got to here:after accfb2d call\n");
                                    /*find min and max of array Y*/
for(i=0;i<(N+1)*(N+1);++i){
                    if(Y[i]>maxxij)maxxij=Y[i];
                    if(Y[i]<minxij)minxij=Y[i];

}
                                    /*output square N by N array*/
for(i=0;i<N;++i){
        for(j=0;j<N;++j){
        ucharxij=(unsigned char)(( (Y[i*(N+1)+j]-minxij) /(maxxij-minxij))*max_pix_val);
        fwrite(&(ucharxij),1,1,outptr1);
            }
    fprintf(outptr1, "%4d ",(int)ucharxij);

}

farfree((float far *)Y);
fclose(outptr1);

}/*end main*/
```

//

```c
void AccFB2d( float huge *X,int maxlevel,float D_image,float H,unsigned int seed)
{
/*      float huge *X; stores a doubly indexed real array of size (N+1)*(N+1)
        int maxlevel; maximal # of recursions,N=2^maxlevel
        float D_image;//width of the image
        float H; parameter H determines fractal dimension D=3-H
        unsigned int seed; seed value for random number generator */

        int i,N,stage;
        float sigma_c,sigma_d,sigma_eps,sigma_eta;//real holding standard deviations
        float a,scale_factor;
        int x,y,y0,D,d;//integer array indexing variables

        srand(seed);
        N=pow(2,maxlevel);                              /*set the initial random corners*/

        printf("we are before sigma_c\n");
        sigma_c=(float)sqrt(2.-pow(2.,(double)H));
        sigma_d=(float)(2.*sqrt(pow(2.,(double)H)-1.));
//      printf("delta= %f N= %d\n",delta,N);
        printf("we are after sigma_c\n");
        a=0.5*sigma_d*gaussian(0.,1.);
        X[0]=sigma_c*gaussian(0.,1.)+a;
        X[N*(N+1)+N]=sigma_c*gaussian(0.,1.)-a;
        a=0.5*sigma_d*gaussian(0.,1.);
        X[N]=sigma_c*gaussian(0.,1.)+a;
        X[N*(N+1)]=sigma_c*gaussian(0.,1.)-a;
```

163

```c
D=N;
d=N/2;
printf("just before assign sigma_eps\n");
sigma_eps=sqrt(pow(2.,1.-H)-pow(2.,-2.+H)-.5)/pow(.5,.5*H);
sigma_eta=sqrt(pow(2.,1.-2.*H)-.5)/pow(.5,(double)H);
    printf("we got to here:before stage loop\n");

for(stage=1;stage<=maxlevel;++stage){

                    /*going from grid type I to type II*/

        sigma_eps *= pow(.5,.5*H);

                    /*interpolate and offset points*/

        for(x=d;x<=N-d;x+=D){

                    for(y=d;y<=N-d;y+=D){

X[x*(N+1)+y]=f4(sigma_eps,X[(x+d)*(N+1)+y+d],X[(x+d)*(N+1)+y-d],X[(x-d)*(N+1)+y+d],X[(x-d)*(N+1)+y-d]);

                    }

                /*going from grid type II to type I*/
                /*interpolate and offset boundary grid points*/

        sigma_eta *= pow(.5,(double)H);
        for(x=d;x<=N-d;x+=D){

            X[x*(N+1)] = f2(sigma_eta,X[(x+d)*(N+1)],X[(x-d)*(N+1)]);
            X[x*(N+1)+N] = f2(sigma_eta,X[(x+d*(N+1)+N],X[(x-d)*(N+1)+N]);
            X[x] = f2(sigma_eta,X[x+d],X[x-d]);
            X[N*(N+1)+x] = f2(sigma_eta,X[N*(N+1)+x+d],X[N*(N+1)+x-d]);

        }/*end x for */
```

```c
                    /*interpolate and offset interior grid points*/
                    sigma_eps *= pow(.5,.5*H);
                    for(x=d;x<=N-d;x+=D){
                        for(y=D;y<=N-d;y+=D){
                            X[x*(N+1)+y] =
f4(sigma_eps,X[x*(N+1)+y+d],X[x*(N+1)+y-d],X[(x+d)*(N+1)+y],X[(x-d)*(N+1)+y]);
                        }
                    }/*end x for*/
                    for(x=D;x<=N-d;x+=D){
                        for(y=d;y<=N-d;y+=D){
                            X[x*(N+1)+y] =
f4(sigma_eps,X[x*(N+1)+y+d],X[x*(N+1)+y-d],X[(x+d)*(N+1)+y],X[(x-d)*(N+1)+y]);
                        }
                    }/*end x for*/

                    D /=2;
                    d /=2;

                }/*end stage for loop*/
                /*scale the computed image to "real" size of image*/
                scale_factor=pow(D_image,(double)H);
                for(x=0;x<=(N+1)*(N+1);++x){
                    X[x] *= scale_factor;
                }

}/*end AccFB2d*/

float gaussian(float mean,float stdev)
//Before this function is called rand needs to be initialized by srand
```

```
    float u,v,x;
    double pi=3.14159265358979;
    u=(float)((float)rand()/(float)RAND_MAX);
    if(u==0.)u=.000000001;
    v=(float)((float)rand()/(float)RAND_MAX);
    if(v==0.)v=.000000001;
//  printf("\nu=%f, v=%e stdev= %f mean=%f\n",u,v,stdev,mean);
    x=(float)(sqrt(-2.*log((double)u))*cos(2.*pi*(double)v)*stdev)+mean;
    return x;
}

float f4(float delta,float x0,float x1,float x2,float x3){
    return((x0+x1+x2+x3)/4.+delta*gaussian(0.,1.));
}

float f2(float delta,float x0,float x1){
    return((x0+x1)/2.+delta*gaussian(0.,1.));
```

Appendix J

Discussion of Fractal Dimensions

There are several points which must be considered under the topic of "fractal analysis" and/or synthetic scene generation using a "fractal seed". These can be summarized to three main categories:

1. Underlying Model: Is the texture monofractal or multifractal?
2. Metrics: Which tool does one use to measure the "dimension" (e.g., power spectrum or variogram)?
3. Generation/Simulation Mechanism: Which synthetic "scene" tool to use (e.g., midpoint displacement or "accurate fractal")?

For the purposes of the study reported here, we assumed a monofractal model and used the power spectrum as the tool for determining the "fractal dimension" of the texture. We then chose the midpoint displacement algorithm as the fractal "scene" generation tool. As part of the initial study, we also applied another scene generation technique (Stoksik, 1995) as it was suggested that it could more accurately replicate a texture whose fractal dimension were the same as the "input" dimension -- this part of the effort was undertaken after the main study was already begun; hence, it was included only in appendices to the main report (Appendix A: Modified Mid-Point Displacement Algorithm) and even then, not in as complete a manner as the other techniques. The approach taken in this report has left some unanswered questions which it was felt should be addressed even if they could not be answered. What instigated this discussion is that the "measured" fractal dimensions of some of the texture maps were outside the accepted bounds for that metric; i.e., fractal dimensions greater that 3 were obtained for some surfaces. *Post facto* efforts to rationalize these results have arrived at some conclusions which are of interest. These will be addressed below, briefly.

First, there are techniques to determine whether or not the object of study is a fractal -- we chose to not use them at the outset for several reasons:

1. limited resources,
2. an incomplete understanding, on our part, of this whole fractal business, and
3. more than likely, there are no accepted procedures to follow anyway (Tsonis and Elsner, 1995)

Some of these are addressed in a paper by Lovejoy et al. (1995). For example, common techniques to measure the fractal dimension of a "process" use variogram or spectral methods to arrive at a "fractal dimension" -- according to Lovejoy et al.(1995), these methods "...measure the scaling exponent of the second order moments...which is not a fractal dimension" unless monofractality is assumed and, indeed, is true. Otherwise, the measurement is erroneous. Many making "fractal measurements" discuss complex techniques; but then, choose those ways of doing things with which they are most comfortable and which are in many cases the simplest (we were no different in that we chose techniques that were already in use by us and with which we were already comfortable). These simple techniques may work for truly fractal phenomena; however, when the process is multifractal, much more complicated techniques are required. Those who shun power spectral methods because of their complexity will find multifractal analysis much more difficult to both accomplish and understand. "A fractal is a geometrical set of points; a multifractal is a mathematical measure. ...many results derived for fractal sets and monofractal functions will not apply (Lavallée et al., 1993)." So, were the effort described in this report to be done again, it is highly recommended that this issue between fractal and multifractal, and even between fractal and Euclidean, be resolved first. The point here is that if our target texture were non-fractal or even multifractal, then the results of applying our metrics have no meaning and the validity of the number used by the synthetic scene generation algorithm would then be suspect.

Second is the problem of the metrics. It turns out that there are a large number of metrics available to determine the fractal dimension of an "object" of study -- some are more or less appropriate depending on the object while others are not and still others are misunderstood (e.g., see Kinsner, 1994). For example, there is the subject of using the power spectrum as a tool to determine the fractal dimension of a surface -- Voss (1988) gives equations for and discusses the use of the 1-, 2-, and 3-dimensional power spectrum for the determination of the fractal dimension of those "objects": a line, a surface, and a volume, respectively. He describes the relationship between them with the conclusion on many analysts' part that one can determine the power spectrum of a vertical cut through a surface and that the slope of this is just one less than the power

spectrum of the surface (2-dimensional) -- it may be that this conclusion is erroneous (Clarke and Schweizer, 1991)) and that the correct relationship between the fractal dimension of a surface and a cut through that surface is for a horizontal cut -- not a vertical cut! This then begs the question of how one determines the power spectrum of a line which forms the horizontal cut or is it truly acceptable to compare results from vertical cuts with 2-dimensional power spectra? Huang and Turcotte (1990) and Turcotte (1992) seem to arrive at acceptable results this way -- they describe the results of applying power spectra to small portions of the topographic map of Oregon with good results in the comparison between one-dimensional vertical cut and two-dimensional spectra (though, in an earlier paper (Huang and Turcotte, 1989) which studied the state of Arizona, the agreement was not very good and they do not discuss the discrepancy, anywhere!). Lam and De Cola (1993) describe an isarithm technique for dealing with the horizontal cut as well as a variogram technique for dealing with the whole surface. We have applied the variogram technique to several of our texture maps for the purpose of measuring with another technique. This is discussed next.

The variogram technique which we applied to 11 each midpoint displacement and accurate fractal maps does not seem to clear up the measurement problem and, in fact, may not even be appropriate (Lavallée et al., 1993) Regardless, figures 1 and 2 show examples of the variogram for two different texture maps, one each for the two fractal generation techniques. The least squares fit to the variogram is supposed to yield the "Hurst Parameter", H. For all of the variograms used here, we fitted only over the "linear" portion with the following results as given in Table 1. So, we have a paradox: of the two simulation techniques, the one which is the most "accurate", performs poorest. At least, this is so for the variogram measurement algorithm; perhaps, another metric would yield different results and, the sample size may be too small to derive a valid conclusion on this point, anyway. According to Tsonis and Elsner (1995), these techniques are inadequate to demonstrate scaling -- what we have done is assumed scaling and not checked the viability of the assumption.
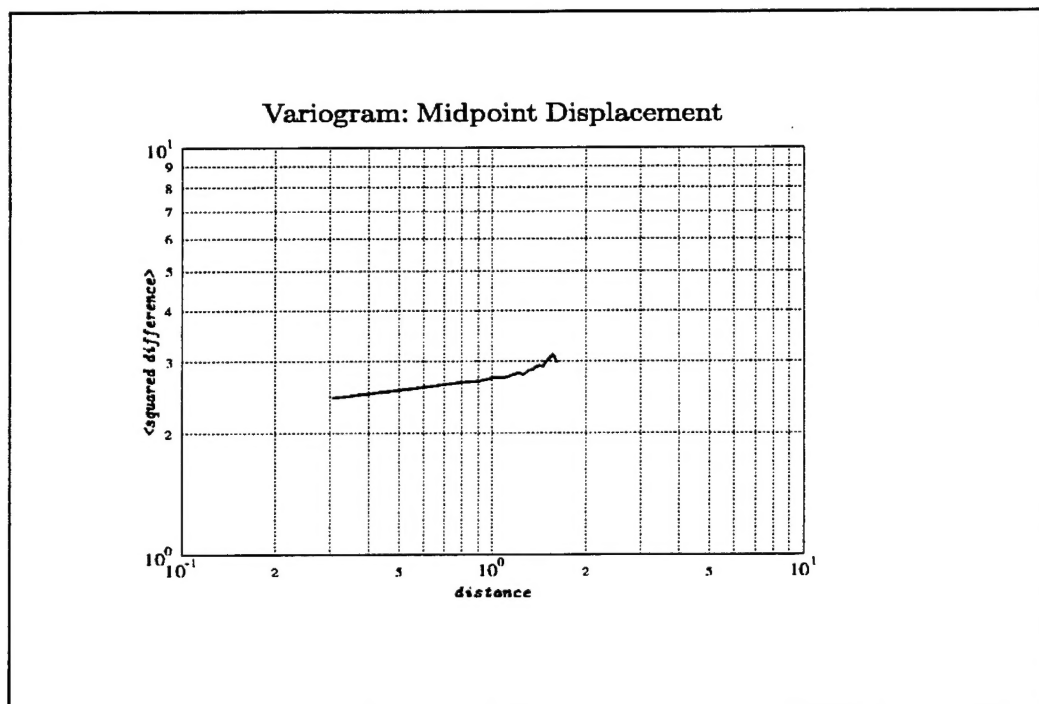
Figure 1J. Variogram estimate for one of the midpoint displacement texture maps.
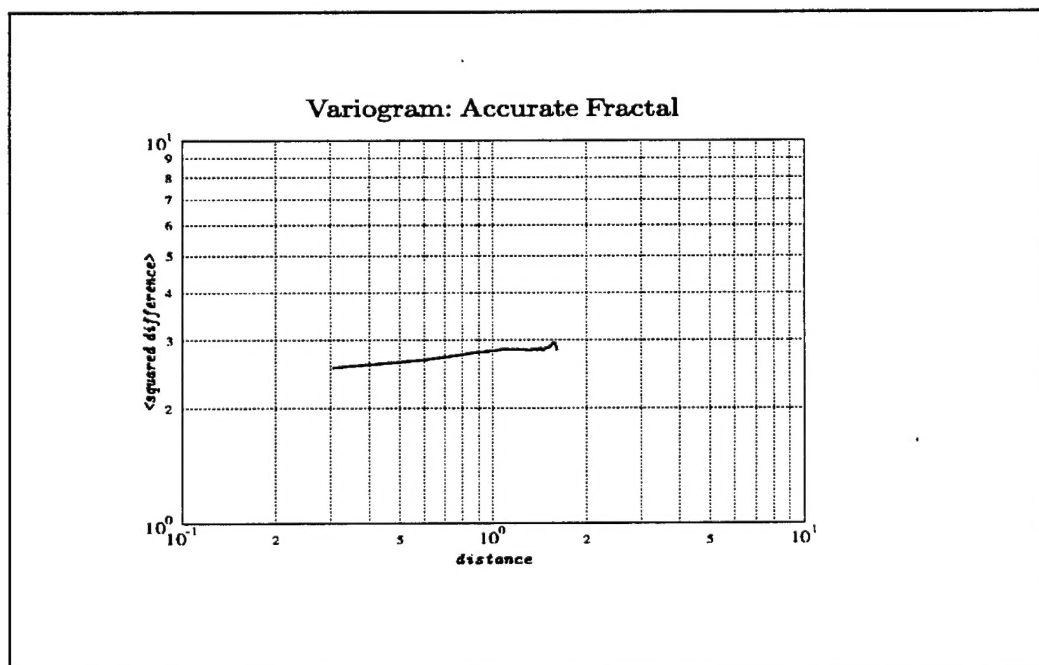

Figure 2J. Variogram estimate for one of the accurate fractal texture maps.

171

Still another issue, also discussed in the paper by Tsonis and Elsner (1995) and others, is that for the type of data with which we are working, self-affine processes are important. This also means that the dimensions that we arrive at are supposed to be the result of an "ensemble average" -- the sample that we are working with may or may not be representative of this process. Similarly, the synthetic generation processes need to be run many times to arrive at a statistically significant sample size (Tsonis and Elsner, 1995, in their experiment ran their sample size to 1000).

Table 1. Variogram measurement of the fractal dimension for 11 texture maps for each of the two different "fractal" texture generation algorithms (the algorithms were run with H=0.28 as a seed).

Source	Hurst parameter (average)	Hurst parameter (standard deviation)	Fractal dimension (D=3-H)
Midpoint displacement	0.286	0.064	2.714
Accurate fractal	0.200	0.066	2.800

As is seen by the results shown in Table 1, different scene generation techniques yield different results -- which one is correct? Or Best? Again, more work is indicated. However, the problem which will remain is one of separating the measurement technique from the process being measured.

Distribution

	Copies
ARMY RESEARCH LABORATORY AMSRL IS BATTLEFIELD ENVIR DIV 2800 POWDER MILL ROAD ADELPHI MD 20783 1145	1
DTIC 8725 JOHN J. KINGMAN RD STE 0944 FT BELVOIR VA 22060 6218	1
SCIENCE AND TECHNOLOGY CORP 555 S. TELSHOR LAS CRUCES, NM 88011	5
US ARMY COLD REGIONS RESEARCH LAB ATTN DR A KOENIG 72 LYME ROAD HANOVER NH 03755	1
US ARMY RESEARCH LABORATORY ATTN MAX BLEIWEISS AMSRL IS EW WSMR NM 88002	20
Record Copy	1
TOTAL	29